On Heterogeneous Network-on-Chip Design Based on Constraint Programming

Ayhan Demiriz
Sakarya University,
Sakarya, 54187, TURKEY
ademiriz@gmail.com

Nader Bagherzadeh
University of California Irvine,
Irvine, CA, 92697, USA
nader@uci.edu

ABSTRACT

Core mapping and application scheduling problems coupled with routing schemes are essential design considerations for an efficient Network-on-Chip (NoC) design. This paper discusses heterogeneous NoC design from a Constraint Programming (CP) perspective using a two-stage solution. Given a Communication Task Graph (CTG) and subsequent task assignments for cores, cores are allocated to the best possible places on the chip in the first stage in order to minimize the overall communication cost among cores. We then solve the application scheduling problem in the second stage to determine the optimum core types from a list of technological alternatives and to minimize the makespan i.e. time to complete all tasks. As a design extension, surface area constraint can be introduced to the underlying problem. The paper reports results based on real benchmark datasets from the literature.

Categories and Subject Descriptors

F.2.2 [**Nonnumerical Algorithms and Problems**]: Sequencing and scheduling; G.1.6 [**Optimization**]: Constrained optimization; F.4.1 [**Mathematical Logic**]: Logic and constraint programming

1. INTRODUCTION

NoC design paradigm has been proposed to overcome traditional chip design limitations that have become increasingly eminent under the reality of single chips made of large number of cores. Such technology requires solutions to challenging general problems as highlighted in [6]: application modeling and optimization; NoC communication architecture and optimization; NoC communication architecture evaluation; and NoC design validation and synthesis. Designing an NoC system requires handling persistent optimization that often demands finding the best tradeoff between conflicting objectives and constraints.

Heterogeneous designs may be more desirable for their utility to run various types of computational tasks efficiently by having cores that can dynamically adjust their voltages (i.e. dynamic voltage-frequency scaling - DVFS problem) and/or can be set to run at certain pre-optimized voltage levels (i.e. voltage-frequency island - VFI problem) to use more efficiently the resources available such as power, area/surface and bandwidth etc. [6, 8]. Core mapping and application scheduling [5] problems are essential optimization problems for designing NoC systems whether they are homogeneous or heterogeneous [9]. In our earlier work [3], a Constraint Programming (CP) based approach was proposed successfully to determine optimum core mapping and application scheduling at transmitted packet level. The proposed model in [3], CPNoC was designed to carry out the optimization task for the homogeneous NoC systems.

This paper discusses heterogeneous NoC design from a Constraint Programming (CP) perspective using a two-stage solution as in [3]. Given a Communication Task Graph (CTG) and subsequent task assignments for the cores, CPU cores are allocated to the best possible places on the chip in order to minimize the overall communication cost among cores. Then, the application scheduling stage is run to determine the optimum core types from a list of technological alternatives and to minimize the makespan i.e. time to complete all computation tasks on CTG. Heterogeneous designs may involve optimization problems that have conflicting terms in their objective functions. To accommodate solutions for the heterogeneous designs, our formulation of CPNoC in [3] is extended in this paper to have a multi-objective function, constraints and new decision variables to determine the best core composition i.e. percentages of various core types on the chip.

The rest of this paper is organized as follows. In Section 2, the proposed method and underlying CP model are presented. Section 3 gives the experimental results on real benchmark datasets from [4]. In this section, we first present results for homogeneous architecture then results from heterogeneous architecture are presented. In our experiments, both 4×4 and 8×8 architectures are utilized. Section 4 introduces the related literature on heterogeneous NoC designs. The paper is concluded in Section 5.

2. PROPOSED OPTIMIZATION MODEL

Assuming that a set of cores is organized as a 2-D mesh of dimensions $n = m \times m$, each core can be labeled according to its position on the mesh (as an x and y coordinate pair) and has the capability of executing several tasks of an application in tandem. The buffer size is assumed to be unconstrained (i.e. infinite buffer size). Any communication link can only

NoCArc'13, December 08 2013, Davis, CA, USA

http://dx.doi.org/10.1145/2536522.2536528.

be occupied by a single packet at any given time without any limit on the bandwidth size. The communication links are bi-directional. In other words, any particular link between two routers can be considered as a separate link (resource) in each direction.

Given a CTG, $G(\mathcal{T}, E)$ where E represents the communication requirements and precedence constraints for the tasks $\mathcal{T}$ with corresponding computation times, core mapping and scheduling problems are tackled in order to assign cores to optimum locations on the target network topology and then to schedule tasks to minimize the makespan. The first part of the problem solved in Stage I is an instance of the Quadratic Assignment Problem (QAP) and can be formulated as a CP model as shown in Eq. 1 (see [3]).

$$\begin{aligned} &\underset{\pi}{\text{minimize}} \sum_{i,j=1}^{n} f_{ij} d_{\pi_i \pi_j} \\ &\text{subject to} \\ &\quad \text{alldifferent}(\pi_1, \ldots, \pi_n), \\ &\quad \pi_i \in \{1, \ldots, n\},\ i = 1, \ldots, n. \end{aligned} \tag{1}$$

where f and d are flow and distance matrices (parameters) respectively. In other words, the parameter d is the Manhattan distance (cost) between all the possible pair combinations of the cores on the mesh and it is the cost of transferring data among cores which is a function of the routing algorithm. Deterministic routing algorithms can easily be incorporated to the optimization problem on hand and can help in finding the optimal application task schedule by utilizing d. The transfer cost based on the XY routing algorithm is proportional to the Manhattan distance which can be calculated between points a and b on a grid as: $\text{TC}_{ab} = |x_a - x_b| + |y_a - y_b|$. Parameter f represents the amount of data to be transferred between each core which is given as part of CTG. Considering all these parameters, the objective function in Eq. 1 can be considered as a representation of dynamic power. In Eq. 1, permutation variables $\pi_i = j$, for $i, j = 1, \ldots, n$. correspond to the location of cores. `alldifferent` constraint enforces unique assignment for each core. In other words, each of the n variables should be assigned a unique value between 1 and n.

Stage I of our implementation can be conducted as in our early work [3] by solving Eq. 1. The optimal solution found in Stage I can be considered as the best floor-planning assignment that does not consider any resource and task precedence constraints. Resources in this context could be considered as bandwidth, power, surface area as well as temperature. Such constraints can be introduced to the optimization problem in the second stage that involves task scheduling with routing.

As in [3], a conventional objective function for the second stage can be represented as following:

$$\underset{t_i \in \mathcal{T}}{\text{minimize}} \quad \max(\text{EndOf}(\, t_i\,)) \tag{2}$$

where $\text{EndOf}(\, t_i\,)$ represents the completion time of task $t_i \in \mathcal{T}$. Considering the variable types available in CP related to modeling the scheduling problems, we can incorporate `interval` and `sequence` variables in our formulation of Stage II. Basically, tasks to be scheduled can be represented as `interval` variables. Therefore, they are defined on timeline and they are associated with starting and end times which are practically continuous values. The `sequence` variables, on the other hand, can be utilized to model (finite) resources (e.g. communication channels that transfer data) and they are coupled with related jobs (e.g. transferring the data). To complete our model, we need to incorporate precedence constraint with suitable CP modeling tools such as `endBeforeStart`, `endBeforeEnd`, `endAtStart`, and `endAtEnd` constraints. To give an example, `endBeforeStart` constraint requires a job t_i to end before the other job t_j starts. Thus, job t_i has been completed before job t_j is started. In order to model (finite) resources by utilizing sequence variables in CP modeling paradigm, we need to associate jobs to be scheduled with (finite) resources (i.e. sequence variables). Considering all these CP modeling tools, we can construct the following CP model to solve scheduling problem as follows:

$$\begin{aligned} &\underset{t_i \in \mathcal{T}, p_i \in P, \ell \in L}{\text{minimize}} \max(\text{EndOf}(\, t_i\,)) \\ &\text{subject to} \\ &\quad \text{sizeOf}(\, t_i\,) = \text{JD}(\, t_i\,),\ \text{forall}\ t_i \in \mathcal{T} \\ &\quad \text{endBeforeStart}(\, t_i, t_j\,),\ \text{forall}\ t_i \ll t_j \in \mathcal{T} \\ &\quad \text{endBeforeStart}(\, p_i, p_j\,),\ \text{forall}\ p_i \ll p_j \in P \\ &\quad \text{endBeforeStart}(\, t_i, p_j\,),\ \text{forall}\ t_i \ll p_j \in \mathcal{T}, P \\ &\quad \text{endBeforeStart}(\, p_j, t_i\,),\ \text{forall}\ p_j \ll t_i \in \mathcal{T}, P \\ &\quad \text{noOverlap}(\, L\,). \end{aligned} \tag{3}$$

where $\ll$ is precedence operator that indicates precedence relation, interval variables $t_i \in \mathcal{T}$ and $p_i \in P$ represent respectively computation tasks given in CTG ($\mathcal{T}$ depicts the set of all tasks) and data transfer jobs (packet transmission) that are generated based on the results of core mapping in Stage I and the underlying routing scheme (XY routing with Wormhole switching in our implementation). `sizeOf` (length) constraint enforces computation task times to be the same as the job durations, JD is given to optimization model as a parameter (i.e. constant) in clock cycles and is determined according to both CTG and the architecture. In addition to precedence constraints constructed based on CTG for computational tasks, there need to be constraints for coupling computational tasks and communication jobs. Basically, data transmission to the next computation node can only start after the current task has been completed. The next computation task can only start after all the necessary data transfer has been completed. The sequence variables, $\ell \in L$, are used for representing the bi-directional data channels that can serve only single flits at any given time. All these particular variables should be associated with related packet transmission jobs in P. `noOverlap` constraint in Eq. 3 practically enforces an orderly job sequencing (packet transmission) of data channels. Therefore, none of the data transmission jobs at flit level overlaps with any other data transmission job on the same data channel excluding opposite directional flows.

The problem aforementioned in Eq. 3 is valid for a homogeneous architecture as computation times of tasks at all cores are considered as same across the chip. One way of modeling heterogeneous architecture is to introduce binary decision variables for each core and technologically alternative core types (e.g. FPGAs, ASICs and GPUs [2],[1],[11]). Assuming that there are n_c different core types, we can introduce $n \times n_c$ binary decision variables into scheduling model given in Eq. 3 to determine optimum core composition. Notice that any solution would prefer fastest cores by default if the objective function in Eq. 3 is not changed. Even considering only the budget constraint, it may not be feasible to choose all of the cores from the fastest available technol-

ogy. Therefore, we need to introduce a penalty term to the objective function for the alternative core types as follows.

$$\begin{aligned}
&\underset{t_i\in\mathcal{T},p_i\in P,\ell\in L,q\in Q}{\text{minimize}} \ \frac{\max(\text{EndOf}(\,t_i\,))}{\lambda} + \sum_{k=2}^{n_c}\frac{\sigma_k}{n}\sum_{j=1}^{n} q_{jk} \\
&\text{subject to} \\
&\quad \sum_{k=1}^{n_c} q_{jk} = 1,\ j = 1,\ldots,n \\
&\quad \text{sizeOf}(\,t_i\,) = \sum_{k=1}^{n_c}\alpha_k \sum_{j=1}^{n} q_{jk}\,\text{JD}(\,t_i\,),\ \text{forall}\ t_i \in \mathcal{T} \\
&\quad \text{endBeforeStart}(\,t_i, t_j\,),\ \text{forall}\ t_i \ll t_j \in \mathcal{T} \\
&\quad \text{endBeforeStart}(\,p_i, p_j\,),\ \text{forall}\ p_i \ll p_j \in P \\
&\quad \text{endBeforeStart}(\,t_i, p_j\,),\ \text{forall}\ t_i \ll p_j \in \mathcal{T}, P \\
&\quad \text{endBeforeStart}(\,p_j, t_i\,),\ \text{forall}\ p_j \ll t_i \in \mathcal{T}, P \\
&\quad \text{noOverlap}(\,L\,) \\
&\quad q_{jk} \in \{0,1\},\ j = 1,\ldots,n;\ k = 1,\ldots,n_c.
\end{aligned} \tag{4}$$

where λ is a normalization coefficient which can be set to the objective value of a solution to Eq. 3, σ_ks are weighting coefficients between 0 and 1 that adjust the preference of certain technologies to others. Notice that q_{jk} is a binary decision variable and takes a value of 1 if kth technological alternative is chosen for the jth core otherwise it is 0. The objective function of Eq. 4 is a weighted combination of makespan and core compositions i.e. percentages of various types of cores. Since there are n_c technological alternatives, one of them is linearly dependent to the rest. Alternative 1 can be arbitrarily chosen as dependent, for the sake of argument. Therefore, the index of technological alternatives (k) for decision variable q_{jk} starts at two. This objective function is a form of multi-objective function. Moreover, this multi-objective function enables optimizing the conflicting terms by finding the best tradeoff between normalized makespan and the core composition. Note that it may be undesirable to prefer faster cores in order to reduce the makespan from some design perspectives such as budget, space and power.

First constraint in Eq. 4 enables the assignment of only one core type to a single place (unit) on chip. Second constraint determines the computation time of the task t_i based on coefficients α_ks. In other words, depending on technological properties of core types (α_k), computation times may vary (below or above) from a baseline time (JD(t_i)). This is inline with the results from [1, 11] since the speeds of various alternative technologies can be summarized as a function of power, and area. The remaining constraints are the same as in Eq. 3 except binary value constraints for the decision variables q_{jk}.

3. APPLICATION ON REAL BENCHMARK DATASETS

We have employed real application benchmark datasets to evaluate the mapping and the scheduling algorithm in this section. Multi-Constraint System-Level (MCSL) benchmark suite [4] provides a set of real applications where each application composes multiple tasks and traffic data patterns between these tasks. MCSL benchmark records the data traffic for different mesh network sizes and measures the execution time for each task in the application. Most of the architectural settings are borrowed from [3], exceptions are specified as needed. Results from both homogeneous and heterogeneous architectures are presented in this section. The CP models are implemented by using IBM ILOG OPL Studio, which is available free of charge to the academicians at IBM Academic Initiative web site at `http://tinyurl.com/cu5txlg`.

Table 1: MCSL Benchmark Suite Applications

Application	Number of Tasks	Number of Comm. Links
R-S code encoder	248	328
R-S code decoder	278	390
ROBOT	88	131
SPEC95 FPPPP	334	1145
SPARSE	96	67
H.264 video decoder	2311	3461

Table 1 shows the applications provided by MCSL which were used as data sets of our mapping and scheduling algorithms. Table 1 also shows the number of tasks for each application as well as the number of communication links. The files obtained by MCSL benchmark include task execution times, the details of communication links between task, and the amount of data for each communication link. The execution time is represented by clock cycles while the data is represented by number of words on each link. Words are 32-bit wide which corresponds to one flit. Each packet contains one header flit and eight data body flits. Between two network nodes, header flits require three clock cycles while a data flit requires only one clock cycle.

For each different application in MCSL Benchmark, we generated 10 different random sets (all the model and data files are available at `http://tinyurl.com/cjseuuz`) of the execution times and the traffic patterns according to distributional parameters provided in the benchmark data specifically the files with 'STP' extension [4]. Two different sizes of the mesh architecture were utilized in our experiments: 4×4 and 8×8. The packet size was set to be eight for all the applications except H.264 which was set to be 64 due to the computational complexity caused by the small packet size in Stage II.

3.1 Homogeneous Architecture

Table 2 presents the results from experiments on 4×4 mesh architecture by solving Eq. 3. All the results were averaged over ten different random realizations and standard deviations were also reported after the ± operator. The total CPU time limit was set to be 1000 seconds in all the experiments for the application mapping model (i.e. Stage I) except H.264 which was set to 3600 seconds. We set the time limit to 1600 seconds in Stage II (i.e. the application scheduling model). We reported the objective values and number of different feasible solutions in both Stage I and Stage II CP models. None of Stage I models (i.e. solving Eq. 1) resulted in an optimal solution. Therefore, reported results are based on the best solutions found until the run time limit was reached. However, all Stage II models produced the optimal scheduling except for the R-S32DEC and H.264 applications. We also reported CPU times for Stage II which differ from the run time limits as most scheduling problems were solved to the optimality within seconds. The most challenging problem was the application scheduling part of the H.264 benchmark. To give an idea about the complexity, a representative problem might have over 67k variables and over 79k constraints for the H.264 benchmark

Table 2: MCSL 4x4 Mesh Results (Average±Std.Dev.)

	Stage I		Stage II			
Application	Objective	Num.of Sol.	Objective	Num.of Sol.	Run Time(sec.)	Latency
R-S32ENC	1324±0	7±0	1821±18.73	1.6±0.5	0.71±0.26	5±0.01
R-S32DEC	2208±0	9±0	2961±151	3.8±1.8	1600±0	12.37±0.45
ROBOT	10169±120	19±5	92818±1914	1±0	1.59±0.35	7.91±0.07
FPPPP	161021±914	15.7±3.68	85371±5914	1.6±0.7	385±443	149.4±6.8
SPARSE	22206±365	19.5±5	19982±1101	1.2±0.4	4.3±2.24	14.49±3.1
H.264	1501505±3007	19±2.36	19532265±727341	1±0	23335±597	632±90

Table 3: MCSL 8x8 Mesh Results (Average±Std.Dev.)

	Stage I		Stage II			
Application	Objective	Num.of Sol.	Objective	Num.of Sol.	Run Time(sec.)	Latency
R-S32ENC	1380±0	41±0	1752±18.64	2.6±0.5	1.9±0.82	5±0.03
R-S32DEC	4020±0	14±0	2959±150	12.6±1.7	3200±0	17.38±0.82
ROBOT	9808±299	50±9	92452±1871	1±0	1.43±0.25	7.80±0.22
FPPPP	304282±3506	22.8±6.66	85317±5913	1.3±0.7	499±502	196.4±9.4
SPARSE	19125±495	58.1±8.8	20012±1101	1.2±0.4	7.1±3.6	17.73±5.7

data in Stage II. Table 2 also reports the latencies in clock cycle units.

Similarly, Table 3 presents results from experiments on the 8×8 mesh homogeneous architecture. However, the results from H.264 were omitted as the second stage becomes too complex and results in memory problems because there are around 137k variables and 178k constraints. The CPU time limits were set to twice as much as the ones in the 4×4 experiments. Since the CPU time limits were higher, the CP solver was able to find more feasible solutions in the first stage (application mapping). However, the CP solver was not able to guarantee that Stage I results were optimal as for the 4×4 experiments.

A representative sample of the progression of the objective value at Stage I is given in Figure 1. The x-axis represents the number of branches that are generated thus far. Basically, the CP solver was able to sift through approximately 700k branches within CPU time limit which is 2000 seconds for the 8×8 experiments. Each objective value corresponds to a unique solution. At the end of the run, the CP solver reports the best solution found within the CPU time limits.

3.2 Heterogeneous Architecture

Applicability of Eq. 4 has been tested by a new set of experiments in this part of the paper. Again, MCSL benchmark suite has been used in this set of experiments. However, H.264 instance has been completely dropped because of the computational complexities. In contrast to Eq. 3, Eq. 4 has additional parameters such as λ, σ and α to be determined ahead of running optimization.

λ is used for normalizing the objective term related to makespan i.e. the completion time of all tasks on CTG. The parameter σ is used for penalizing composition of the cores. Assuming that one of the core type is more preferable than the rest, we can practically penalize the usage of the remaining core types in the composition of the chip. Notice that σ_k for $k = 2, \ldots, n_c$ should be specified for each technological alternative in our formulation. By using a σ_k closer to 1, we can practically penalize more the usage of that particular core type within the composition of chip. In our experiments, three different technological alternatives have been used. Hypothetically, we can think of these core types as Slow, Regular, and Fast. In our experiments, σ_2 and σ_3 were set to 0.25 and 0.5 respectively. Technically, budgetary constraints can be easily used for determining parameter σ. Parameter α is used for assessing the job duration times of technological alternatives. $\alpha = [1.4, 1, 0.7]$ is chosen for our experiments. Expressly, it takes 1.4 times more for running computation tasks on a slow core type than the regular core type and it takes 30% less time to run tasks on fast core type than the regular core type. Alternatively Job Duration times (JD) could have been picked differently for each task on different core types. However, the parameter α is used for its simplicity.

Figures 3 and 5 report composition of cores yielded by experiments to solve Eq. 4 for 4×4 and 8×8 architectures respectively. These figures practically show percentage of each core type in CPU core composition. Basically, fast cores are preferred for the 4×4 architecture in almost all datasets except R-S32ENC. This indicates that bandwidth is responsive and level of parallelization responds positively in increasing speed of the cores. On the other hand, slow cores are preferred for the 8×8 architecture. This underlines that there is no need to utilize faster cores as the level of parallelization is very high and these cores can wait for the data transmissions to be completed in most cases. Therefore we can conclude that this particular architecture is network-bandwidth sensitive [7]. Table 4 reports completion times in clock cycles for all applications in MCSL benchmark. The results are averaged on 10 realizations of the datasets (applications). The values of parameter λ used in the experiments are also listed in Table 4. Notice that parameter λ is chosen by considering the results given by Tables 2 and 3 except for application FPPPP. As a bandwidth parameter, 64 bit packet size is used for FPPPP to reduce the number of decision variables and constraints for the CP model to solve Eq. 4 - instead of 8 bit used for rest of the applications (see [3] for architectural parameters). It can be observed from Table 4 that completion times are shortened significantly in most cases for both 4×4 and 8×8 architectures. Even if most of the cores are formed by slow ones, the remaining fast cores on the chip can speed up the computational

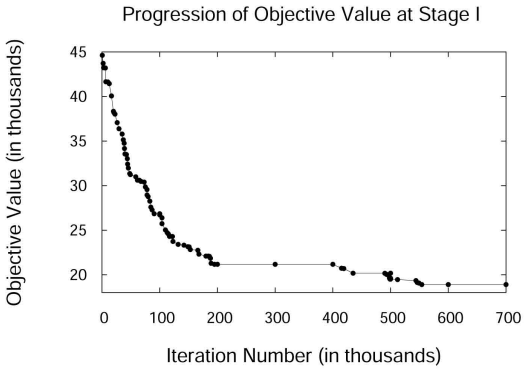

Figure 1: Progression of Objective Value at Stage I of SPARSE Dataset on 8×8 Architecture

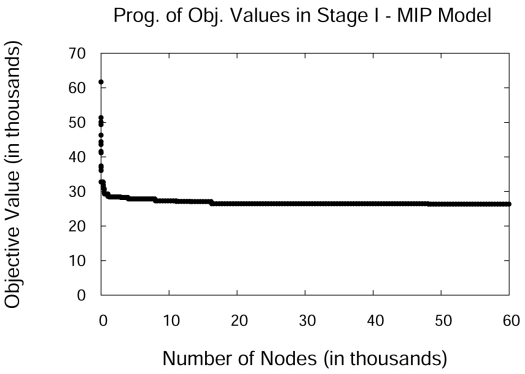

Figure 2: Progression of Object Value of Stage I MIP Model on SPARSE Dataset on 4×4 Architecture

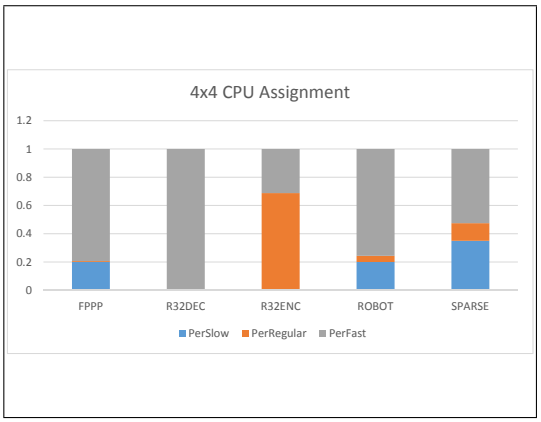

Figure 3: Composition of Core Assignment on 4×4 Architecture

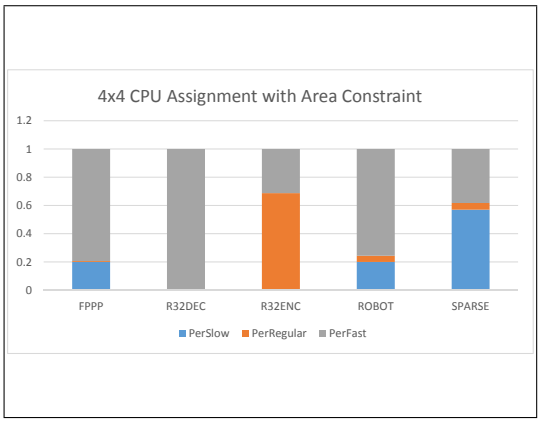

Figure 4: Composition of Core Assignment on 4×4 Architecture with Area Constraint

Table 4: Completion Times in Clock Cycles for Heterogeneous Architectures

		4×4 Completion Times		8×8 Completion Times	
Application	λ	Eq. 4	Eq. 4 w/area	Eq. 4	Eq. 4 w/area
R-S32ENC	1800	1452	1452	1466	1466
R-S32DEC	3000	2138	2138	2151	2151
ROBOT	92000	66675	66675	65082	65057
FPPPP	60000	59853	59853	59783	59773
SPARSE	20000	14290	14259	14294	14289

tasks significantly. According to results from [1] and [11], the functional relationship between cores' speed and area usage can be constructed easily. However, we can introduce a parametric constraint to represent surface/area usage on the chip for simplicity. In most of the design problems, surface/area usage is an important factor and Eq. 5 defines a constraint to consider the area limitation. By picking ω_k for $k = 1, \ldots, n_c$ and Ω appropriately, we can define a form-factor for various core types. $\omega = [0.9, 0.8, 0.65]$ and $\Omega = 0.8$ were chosen in our experiments. Experiments by solving Eq. 4 with area constraint (i.e. Eq. 5) were also conducted on MSCL benchmark datasets. Results are reported in Figures 4 and 6 and Table 4. When compositions of the chips within each architecture are compared, there is almost no change for 4×4 architecture (see Figures 3 and 4). This means that solutions to Eq. 4 already satisfy the area constraint for the 4×4 architecture. However, introducing area constraint changes the solutions for 8×8 architecture.

$$\sum_{k=1}^{n_c} \frac{\omega_k}{n} \sum_{j=1}^{n} q_{jk} \leq \Omega \tag{5}$$

4. RELATED WORK

A two-stage solution to core mapping and application scheduling problems was also proposed in [9]. The solution is reached by running iteratively these two consecutive stages (master and sub-problems). In each iteration, a new cut was introduced to the master problem in order to get closer to the optimal solution and satisfy the feasibility of scheduling. In [9], the master problem (core mapping) is modeled by integer programming and sub-problem (scheduling) is modeled by CP. Since there are no task deadlines in our model, it is always feasible to find a solution to the scheduling problem in our case. On the other hand, our scheduling model is finer-grained than the one proposed in [9].

In contrast to DVFS, many implementations aim to solve the static voltage-frequency island problem to optimize design of heterogeneous NoC systems in order to run special

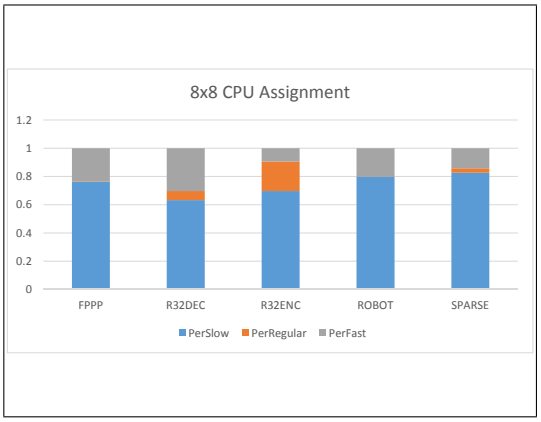

Figure 5: Composition of Core Assignment on 8×8 Architecture

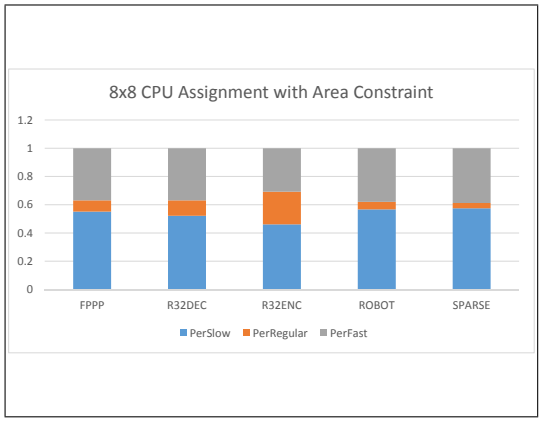

Figure 6: Composition of Core Assignment on 8×8 Architecture with Area Constraint

applications [8]. A heterogeneous NoC design was also proposed in [2] by implementing core mapping as a 2D-packing problem and by finding a heuristic solution to underlying optimization problem. Power usage has also been taken into consideration for scheduling phase in [2]. Network-bandwidth and latency-sensitive designs can be considered for heterogenous systems [7]. Our experimental results indicate that slow cores may be utilized in larger tiles. Latency-sensitive case has been tackled by blending CPU cores and GPUs to solve DVFS problem in [10]. The proposed method in [10] delays some GPU packets by routing through non-minimal paths without compromising the performance.

5. CONCLUDING REMARKS

A CP-based two-stage model is proposed to solve the core mapping and the application scheduling problems for heterogeneous NoC architectures. The major advantage of using CP is the clarity and understandability of the models. We successfully experimented our model on various MCSL benchmark datasets. Surface/area constraint has been introduced to our formulation. It has been practically shown that similar constraints for power, temperature etc. can easily be implemented in CP framework of heterogeneous NoC architectures.

6. REFERENCES

[1] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai. Single-chip heterogeneous computing: Does the future include custom logic, fpgas, and gpgpus? In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '43, pages 225–236, Washington, DC, USA, 2010. IEEE Computer Society.

[2] D. Demirbas, I. Akturk, O. Ozturk, and U. Güdükbay. Application-specific heterogeneous network-on-chip design. *The Computer Journal*, 2013.

[3] A. Demiriz, N. Bagherzadeh, and A. Alhussein. Cpnoc: On using constraint programming in design of network-on-chip architecture. In *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, pages 486–493, 2013.

[4] W. Liu, J. Xu, X. Wu, Y. Ye, X. Wang, W. Zhang, M. Nikdast, and Z. Wang. A noc traffic suite based on real applications. In *VLSI (ISVLSI), 2011 IEEE Computer Society Annual Symposium on*, pages 66–71, 2011.

[5] M. Lombardi and M. Milano. Optimal methods for resource allocation and scheduling: a cross-disciplinary survey. *Constraints*, 17(1):51–85, 2012.

[6] R. Marculescu, Ü. Y. Ogras, L.-S. Peh, N. D. E. Jerger, and Y. V. Hoskote. Outstanding research problems in noc design: System, microarchitecture, and circuit perspectives. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 28(1):3–21, 2009.

[7] A. K. Mishra, O. Mutlu, and C. R. Das. A heterogeneous multiple network-on-chip design: an application-aware approach. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 36:1–36:10, New York, NY, USA, 2013. ACM.

[8] Ü. Y. Ogras, R. Marculescu, D. Marculescu, and E. G. Jung. Design and management of voltage-frequency island partitioned networks-on-chip. *IEEE Trans. VLSI Syst.*, 17(3):330–341, 2009.

[9] M. Ruggiero, D. Bertozzi, L. Benini, M. Milano, and A. Andrei. Reducing the abstraction and optimality gaps in the allocation and scheduling for variable voltage/frequency mpsoc platforms. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(3):378–391, 2009.

[10] J. Yin, P. Zhou, A. Holey, S. S. Sapatnekar, and A. Zhai. Energy-efficient non-minimal path on-chip interconnection network for heterogeneous systems. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, ISLPED '12, pages 57–62, New York, NY, USA, 2012. ACM.

[11] T. Zidenberg, I. Keslassy, and U. Weiser. Multiamdahl: How should i divide my heterogenous chip? *IEEE Computer Architecture Letters*, 11(2):65–68, 2012.