Communication Aware Compiler for Mesh-Structured Reconfigurable Processors on Single/Multi Chip

Yi Lu[1], Qinhao Wang[1], Amir Masoud Gharehbaghi[1] and Masahiro Fujita[2]
[1]Department of Electronic Engineering, The University of Tokyo
[2]VLSI Design and Education Center, The University of Tokyo
2-11-16, Yayoi, Bunkyo-ku, Tokyo, Japan, 113-0032
E-mail : [1]{yilu,wang,amir}@cad.t.u-tokyo.ac.jp, [2]fujita@ee.t.u-tokyo.ac.jp

Abstract: Many-core system performance is still underutilized in many cases. The program optimization on highly parallel systems is hard and usually done manually. The inter-core data transfer delay highly affects the system performance in deep sub-micron age. To overcome these problems, in this paper, we propose an integer linear programming (ILP) based method to analyze and optimize a program running on a mesh-structured processors array. The proposed model includes communication-aware operation binding and mapping as well as data transfer routing. With this flexible ILP based formulation, optimized binding, mapping and routing is determined for a given program on the target architecture. Our ILP based formulation can also be used for high level ECO while performing high level synthesis or targeting multiple chips architecture with two-step ILP.

Keywords— **High-Level Synthesis, Reconfigurable Architectures ,Multi/Many-Core Systems, Integer Linear Programming, Engineering Change Order**

1. Introduction

Reconfigurable many-core systems have been introduced to overcome the power consumption and single-core frequency limit problem. However, programming on such kind of highly parallel system is considered a hard task even expert programmer need carefully schedule their program. In addition, with sub-micron technologies development, the communication delay of inter-core data transfer has become the dominate latency factor in many-core system. If compilers could provide a good mapping and binding of each operation as well as scheduling of data transfer, the many-core system could achieve higher performance and become more program-friendly. The binding and mapping of operation with communication delay is not new, there are many research on hardware synthesis. Such as distributed register (DR) architecture in high level synthesis[1] [2]. However, those kind of architectures cannot be directly used in many-core systems, as the wire and register for communication may be changed during the synthesis flow in DR model. Many-core systems have fixed layout and number of register for communication. On the other hand, we could not use general processor task scheduling methods[3]. As in general processor task scheduling method, the model usually ignores the interconnection delay between processors and the whole processor array usually share same memory space, which could not satisfy our assumption on mesh-structured reconfigurable processor architecture. As a result, in this paper, we extend the DR model to add more constraint on wire and register use. We have proposed an ILP based formulation to solve the aforementioned problems. Moreover, our proposed model can be used for Engineering Change Order (ECO) or targeting multi-chip architectures.

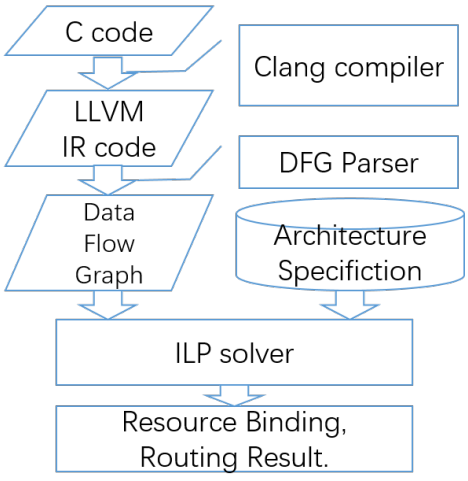

Figure 1. Compiler architecture

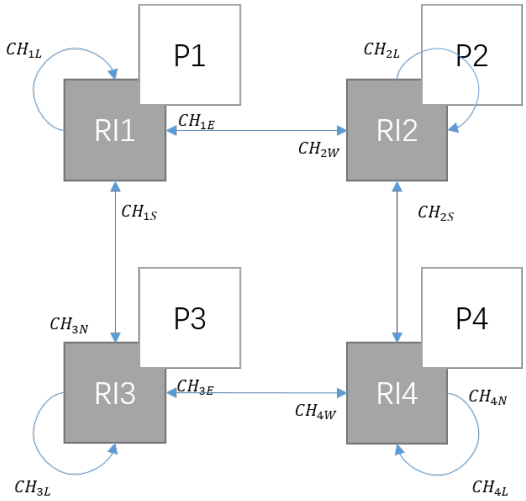

Figure 2. A 2x2 mesh-structured processors array

2. Problem Description

Without loss of generality, in this paper we use C program as the input program. We use Clang[4] to parse the input C program into LLVM intermediate representation code. Then, we

extract the data flow graph (DFG) of the original C program from the LLVM intermediate representation code. Finally, we perform mapping, binding, and data transfer routing of the DFG to the target hardware. The overall compiler architecture is shown in Figure 1. Note that the DFG in our method, may be generated from other programming languages or models.

2.1 Target architecture

Our target architecture is a mesh-structured reconfigurable processors array, this architecture shares the wire resources as well as register resources. As shown in Figure2, the processors are interconnected by the register interface(RI), to simplify our approach, we restrict that each RI could hold only one unit of data for data transfer. As the wire is shared by two neighbor processors, which indicates that two neighbor processors cannot exchange data simultaneously. In above architecture, we name the transfer channel at each processor as $\{N, W, E, S, L\}$, N(orth), W(est), E(ast), S(outh) represent the four directions for data transfer from one processor to its neighbors. L(ocal) means that the data will be located at the processor and can be used for computing and/or transfer in the next cycle. Here we label each processor i as p_i and the communication channel with $ch_{i,dir}$ where $dir \in \{N, W, S, E, L\}$.

2.2 Data flow graph

In this paper, our data flow graph (DFG) used for analysis is generated from C source code. A DFG is a directed graph defined by $G(V, E, \delta)$, where $V = \{v_i | i = 1, 2, , n_{ops}\}$ is the set of nodes of the graph. Each node $v_i \in V$ corresponds to one operation in the program. $E = \{(v_i, v_j) i, j = 1, 2, , n_{ops}\}$ is the set of edges, which represents a data transfer dependency of operations. The δ_i is the latency time for each $v_i \in V$. In this paper, to simplify the model and computing, we assume all the operation latency is one time unit. As an example, Figure3 shows a DFG with 6 operations.

3. Problem Formulation

In this section, we formally present the model of the channel and functional unit binding as well as data transfer problem using integer linear programming (ILP).

3.1 ILP variables

Given a DFG $G(V, E, \delta)$ and a target architecture, we use binary variables $X_{op,t,p}$ to define the operation binding. Here

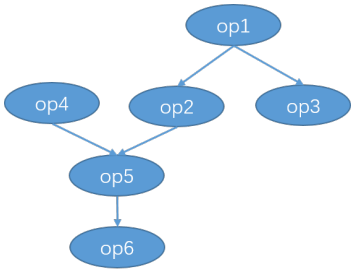

Figure 3. A data flow graph example with 6 operations

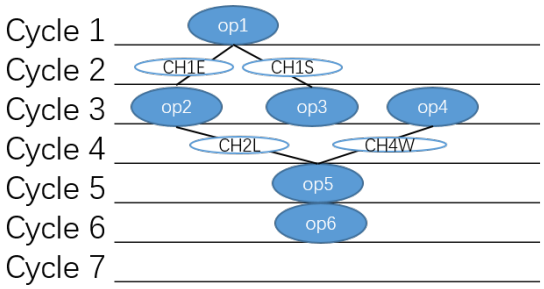

Figure 4. Executing procedure of DFG in Figure3 on a 2x2 architecture

op indicated the operations in DFG, t is the global clock cycle and p is the binded processor and $X_{op,t,p}$ indicates if an operation is binded to a specific time point on a specific processor. To handle the data transfer between the operations, we introduced another binary variables $D_{i,t,ch}$, where i label the data dependency in DFG, t is clock cycle and ch is a channel which is used for transfer data.

3.2 ILP constrains and object function

As we introduced the binary variables to represent the operation binding as well as data transfer, we need appropriate constraints as well as the optimization object to get the result with this model via ILP solver. Basically, we have the resource constraints that every operation should be only bound to one processor at one specific time point:

$$\sum_t \sum_p X_{op,t,p} = 1$$

For each time, at most one operation can be executed on one processor:

$$\forall t, p, \sum_{op} X_{op,t,p} \leq 1$$

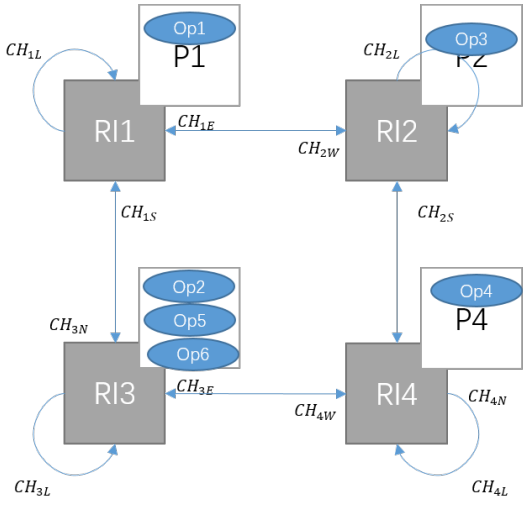

Figure 5. Binding result on DFG in Figure3 on a 2x2 architecture

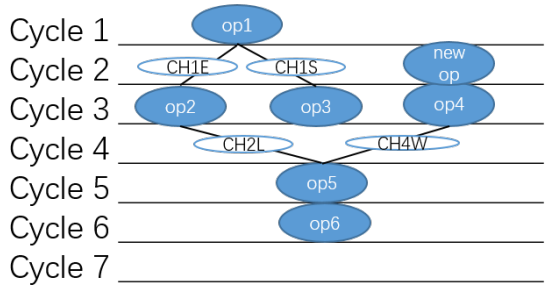

Figure 6. A simple ECO result with add one extra node in DFG on Figure3

For data transfer, only one data transfer could be done in one channel at one time:

$$\forall t, ch, \sum_{i} D_{i,t,ch} \leq 1$$

To correctly handle the wire resource limitation in our data transfer procedure, we also introduce the wire resource constraint in our model. As Figure 5 shows, CH_{1E} and CH_{2W} shared the same wire and the data transfer between processor 1 and 2 cannot be done simultaneously in one cycle, so we iterate all the possible data transfer channels which share the wire resource:

$$\forall t, ch \in \{same\ wire\}, \sum_{i} D_{i,t,ch} \leq 1$$

Also as the operation is related to our data transfer, for each $X_{op,t,p} = 1$ there exists a data requirement $req(D_{i,t-1,ch_{tar-p}})$, indicating that in previous cycle, the required data i should be located in the channel targeted the processor p, and also for the next cycle, it will generate new data i_next for transfer as $gen(D_{i_{next},t+1,ch_p})$:

$$-X_{op,t,p} + \sum_{ch \in req(D_{i,t-1,ch_{tar-p}})} D_{i,t-1,ch} \geq 0$$

$$-X_{op,t,p} + \sum_{ch \in gen(D_{i_{next},t+1,ch_p})} D_{i_{next},t+1,ch} \geq 0$$

In addition, for each time step, the data transfer has its continuity, as a continuity constraint where ch_{tar} is all possible channel for the target processor, similar to the above one:

$$-D_{i,t,ch} + \sum_{ch \in ch_{tar}} D_{i,t+1,ch} + X_{op_i,t+1,p} = 0$$

To reduce the solution space, we also consider precedence constraints between the operations, which mean that a scheduling range can be computed for each operation. This range is an interval between the earliest start time and the latest start time at which the operation can be executed. The earliest start time and the latest start time can be computed by using as soon as possible (ASAP) and as late as possible (ALAP) algorithms respectively.
With these constraints, to get the optimized result, we will evaluate all possible end node in DFG, and find the largest one as the execute time, the object function is:

$$Minimal : Max(t \times X_{op,t,p})$$

To correctly handle this object function, we introduce an auxiliary variable call MAX, and we should have:

$$\forall t, p, \sum_{t} \sum_{p} t \times X_{op,t,p} \leq MAX$$

3.3 ILP based ECO

With this model, we could easily constraint the architecture as well as DFG by change the binary variables. We could keep part of the previous computed operation binding information in variable $X_{op,t,p}$ then start the ILP solver again to perform ECO. As an example in Figure6, we keep all the results on function binding except for node 4 as constraints for our ILP solver, and we run the ILP solver again to get the result, here in this example we keep all the $X_{op,t,p}$ except for node 4 and new node value, re-generate the constraints and use ILP solver to solve them again. If the model is not feasible, we could release some more operation binding after the node and try to re-solve the model. With computed information, the whole formula could be solve faster than without pre-computed information.

3.4 ILP based DFG scheduling on multiple chips

We also try to apply this model in circuit partition problem on multi-chips. To correctly handle the multiple chips mapping, we try two approaches for the DFG scheduling problem on multi-chips. First approach is to re-constraint the whole model, we introduce different delay values at the wire on inter-chips and intra-chip in our model, however the computation complexity is large.
The other approach is to first slice the original DFG into multiple sub DFG. Figure 8 shows an example that we sliced the DFG into two parts program 1 and program 2 and each part shown in a different color. Then we compute their execute latency with our model. Here we could see program 1 and 2 as a new DFG with 2 nodes. As the target chips is also constructed in a mesh way, we could apply our model on this new DFG with inter-chip delay to schedule on the upper level mesh-structured chips array. Here we call this approach a two-step ILP method.

4. Experiments

4.1 Experiments setup

We have implemented the proposed with C++/Linux environment and all the experiments are conducted on a workstation with an Intel Core-i7 2.9GHz CPU and 4GB RAM. Our target architecture is a 2x2 mesh-structured reconfigurable processor array. We use Gurobi[5] as our ILP solver.

4.2 Experiment Results

Table 1 shows the results of mapping some DFGs on a 2x2 architecture, Figure 4,5 shows a detail example with how the

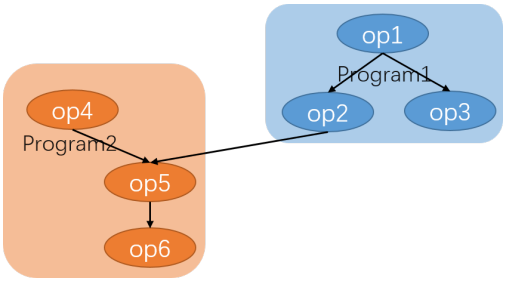

Figure 7. A sliced DFG from Figure3

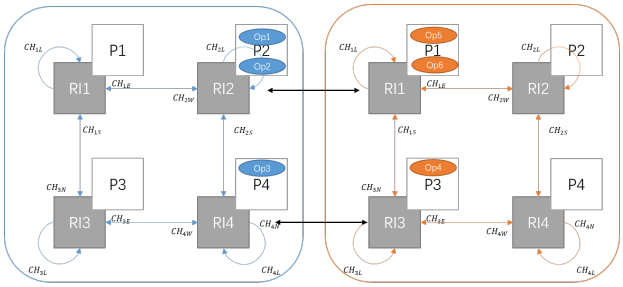

Figure 8. Mapping result of DFG in Figure 7 on two 2x2 mesh-structured reconfigurable processor chips

whole DFG executed on the target architecture. We could see that this model correctly handle both the operation binding as well as data transfer in target architecture. However, as the complexity increases, we could see that the solver run time increases significantly. Note that this model is very flexible, efficient as ILP gives an optimized result for our future research.

To evaluate ECO, we add some more operations into our data flow graph, and we keep part of the computed binding result on our architecture, then we reuse our model to get the incremental result as Table 2 shows. These results indicate that if we could first determine some of the binding of operations, the result searching space will significantly reduced and ILP solver could finish the search quickly.

For multi-chips mapping and scheduling, we introduced a two chips architecture, each chip consists a 2x2 mesh-structured reconfigurable processor array. Here we assume our inter-chip communication latency is 5 clock cycles.

In our experiments, we split the original DFG into two parts manually and apply our method for these two parts on each chip. Table 3 shows some example of multi-chips mapping result. Latency 1 and 2 indicate the individual executing time for the sliced parts of DFG with out considering inter-chip data transfer. However, as there exists data transfer among chips, The final latency will be affected by both individual mapping results as well as the inter-chip data transfer delay. Figure 8 shows the mapping results of DFG in Figure 7. We could see although the individual latency is 3 and 4 clock cycle. After integrate, the total latency becomes 10 clock cycle as the data from operation 2 to operation 5 needs 5 clock cycles to transfer.

Table 1. DFG mapping results

	DFG Node	DFG Edge	Latency	Time(s)
s1	5	4	7	0.85
s2	6	5	7	1.1
filter	16	15	12	170
iir	23	26	16	422
fft	25	24	8	15

Table 2. ECO changing results

	ECO Node	ECO Edge	Latency Before	Latency ECO	Time(s)
s2	1	1	7	7	0.142
filter	3	2	12	13	0.73
iir	6	4	16	18	1.21

Table 3. DFG mapping on 2 Chips with partition

	Latency 1	Latency 2	Total Latency	Time(s)
s2	4	3	10	2
filter	11	13	15	14
iir	12	13	18	33

5. Conclution and Future works

In this paper, we proposed an ILP based method to analyze and optimize a program running on a mesh-structured processors array. This model handles communication-aware operation binding and mapping as well as data transfer routing correctly, we also analyzed how we enhance this model to deal with ECO as well as multiple chips binding problem. These experiment give us some optimized result on the program operations scheduling problem for our future research use.

In the next stage we will expand current model to handle program with pipeline scheduling. As generally this kind of architecture will be used for computing looped programs and we'd like to optimize the throughput for these kind of program. We also interested in finding some heuristic method to solve this problem without losing too much performance in latency.

References

[1] Shanghua G., et al. "Interconnect-Aware Pipeline Synthesis for Array-Based Architectures." *IEICE TRANSACTIONS on Fundamentals of Electronics* Communications and Computer Sciences 92.6 2009.

[2] Huang, Wei-Sheng, et al. "A multicycle communication architecture and synthesis flow for global interconnect resource sharing." *Design Automation Conference, 2008. ASPDAC 2008.* Asia and South Pacific. IEEE, 2008.

[3] Hoang, Phu, and Jan Rabaey. "Hierarchical scheduling of DSP programs onto multiprocessors for maximum throughput." Application Specific Array Processors, 1992. Proceedings of the International Conference on. IEEE, 1992.

[4] http://clang.llvm.org/

[5] http://www.gurobi.com/