Pareto Optimal Scheduling of Synchronous Data Flow Graphs via Parallel Methods*

Yu-Lei Gu[1,2], Xue-Yang Zhu[1], Guangquan Zhang[2]

[1] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[2] School of Computer Science and Technology, Soochow University, Suzhou, China
{guyl, zxy}@ios.ac.cn and gqzhang@suda.edu.cn

Abstract. Synchronous data flow graphs (SDFGs) are widely used to model streaming applications such as multimedia and digital signal processing applications. They usually run on multicore processors and are required a high throughput, which in turn may increase the energy consumption. In this paper, we present a parallel framework to explore the Pareto space of energy consumption and throughput of SDFGs and to find the schedule of each Pareto point. The considered multicore platforms are heterogeneous. We present an exact method pruning the state space according to the properties of SDFGs and two approximate solutions to make the processes faster. Our experimental results show that our methods can deal with large scale models within reasonable execution time, and perform better than the existing methods.

Keywords: Synchronous Data Flow Graphs, Multicore, Pareto optimization, Scheduling, Parallel

1 Introduction and related work

Embedded systems are everywhere today. They are in smart phones, e-book readers, portable media players and digital printers, etc. Streaming applications like audio and video processing, usually modeled by *Synchronous data flow graphs* (SDFGs) [5], are an important class of applications in these electronic devices. Energy efficiency is an essential issue in these devices, for reasons like the increasing demand for portable devices or the heat dissipation.

Streaming applications are usually required to reach a high throughput. The use of heterogeneous multicore processors to improve the throughput of streaming applications has become a feasible solution. However, a higher throughput is usually achieved at the cost of the increase of energy consumption. Designers have to carefully tune the mapping of applications on the platforms to meet performance requirement.

Most mapping methods reported in the literature fall under design-time mapping [7]. The optimization goal of the mapping includes timing, energy consumption and reliability, etc. [6] and [2] present methods to achieve significant energy

* This work is partially supported by 973 program (No. 2014CB340701) and the National Natural Science Foundation of China (Nos. 61472406 and 61472474).

savings. [1] and [4] perform optimization for both energy consumption and execution time. However, these methods only consider homogeneous architectures. [9] works on heterogeneous architecture but only take energy consumption into consideration.

In this paper, we are concerned with constructing throughput and energy efficient static (compile-time) schedules of SDFGs on a heterogeneous multiprocessor platform. For a given platform, even we consider only one optimization criterion, e.g. throughput, the scheduling and mapping problem is already NP-complete [7]. [10] uses model checking to address the same problem and provides exact solutions. In this paper, we try to prune the state space and present a more efficient parallel algorithm which returns exact results. Two approximative methods are provided for larger models.

2 System Model and Problem Formulation

An *execution platform* P is a set of heterogeneous processors. For each processor p, the power consumption is defined by the consumption rates when p is used and when it's idle. The power consumption of processor p_1 shown in Fig. 1(b) is 90 when it's in use, for example.

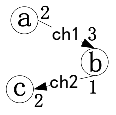
A simple SDFG is depicted in Fig. 1(a). The nodes are called *actors*, modeling the computations of a system. The edges are FIFO channels, transferring the data items, called *tokens*. An essential property of SDFG is that when an actor starts an execution, also called *firing*, it consumes the same amount of tokens from its incoming edges, and when an actor ends a firing, it produces the same amount of tokens to its outgoing edges. The numbers of tokens are called consumption rate and production rate of edges, respectively. They are labeled on each edge. Each actor is weighted with a set of computation times, corresponding to processors. For example, actor a of G_1 in Fig. 1(a) need 1 unit of time to finish on p_1 and 2 units of time on p_2, respectively.

A ***System model*** $\mathcal{M} = (G, P)$ includes an SDFG G and its execution platform P. A simple system model $\mathcal{M}_1$ is shown in Fig. 1(a) and Fig. 1(b).

An SDFG G is *sample rate consistent* [5] if and only if there exists a positive integer vector q. After any sequence of actor firings conforming to q, the number of tokens in the channels are equal to their initial state values. The repetition vector of G_1 is $q = [3, 2, 1]$ for example. An *iteration* is a firing sequence in which each actor α occurs exactly $q(\alpha)$ times. We consider only sample rate consistent SDFGs. Only such SDFGs are meaningful in practice. A *static schedule* arranges computations of an algorithm to be executed repeatedly. An *f-schedule* of system model $\mathcal{M} = (G, P)$ is a static schedule arranging f consecutive iterations of G running on P.

The *throughput* (denoted by *thr*) of f-schedule S is the average number of iterations per unit time, that is, $thr = \frac{f}{T}$, where T is the total execution time of S. The throughput of schedule S_1 shown in Fig. 1(c) is $1/8 = 0.125$ for example. The total energy consumption(denoted by *tec*) of f-schedule S is the sum energy of all processors. For each processor, it includes the energy consumed while it's in idle and in use. The *energy consumption*(denoted by *ec*) of S is $ec = \frac{tec}{f}$. For schedule S_1, for example, $ec = [(2 * 10 + 6 * 90) + (2 * 20 + 6 * 30)]/1 = 780$. A

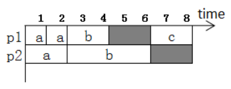
Pareto point is a tuple (thr, ec) of S in which one element (thr or ec) becomes better must make another element worse. Fig. 1(d) shows two Pareto points of the system model $\mathcal{M}_1$, which are $(0.1, 720)$ and $(0.125, 780)$. Schedule S is a *Pareto schedule* when (thr, ec) of S is a *Pareto point.*



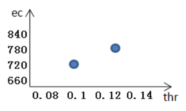
(a)

	power		time		
	inuse	idle	a	b	c
p1	90	10	1	2	2
p2	30	20	2	4	4

(b)



(c)



(d)

Fig. 1. The system model $\mathcal{M}_1$ and its schedules. (a) The SDFG G_1; (b) the execution platform P_1 and the execution time of actors in G_1 on different processors; (c) a periodic schedule S_1 with thr=0.125 and ec=780;(d)Pareto points diagrams of $\mathcal{M}_1$

Given a system model $\mathcal{M} = (G, P)$ and the number of iterations f, the problems we address are to find all Pareto f-schedules.

3 Pareto Optimal Scheduling and Mapping

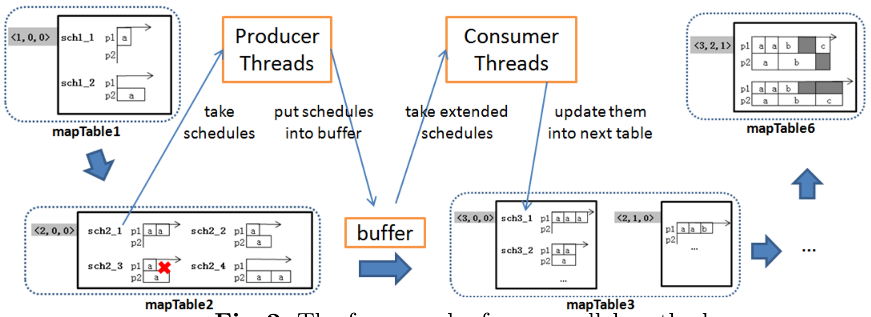


Fig. 2. The framework of our parallel method

We use a parallel framework to find all Pareto f-schedules from a system model. We use a map table to store the constructing schedules at each step. Initially, a constructing schedule includes one firing. The number of firings of schedules in the map table is increased by one at each step until all $f * qSum$ firings have been allocated, where $qSum$ is the sum of elements of the repetition vector q. That means the execution of f iterations of an SDFG has been mapped and scheduled. A map table is created by producer-consumer threads extending its previous map table.

Taking Pareto 1-schedules of $\mathcal{M}_1$ for example, we show our method framework in Fig. 2. The *key* of a map table is a vector contains the scheduled number of firings of each actor. Taking $key < 1, 0, 0 >$ in $mapTable1$ for example, it means actor a has already be executed once while other actors none. The *value* of a map table is a set of constructing schedules. The number of firings of each actor of each schedule equals the corresponding element of the key. Taking $value$ with $key < 1, 0, 0 >$ in $mapTable1$ for example, it contains two constructing schedules, in which a firing of actor a is allocated on $p1$ and $p2$, respectively. We allocate one enable actor on one processor each time, called *extension*, based on the constructing schedules in previous map table. It finally stores all the Pareto schedules when the algorithm finishes. And we finally get two Pareto schedules in the $mapTable6$.

Producer-consumer mode is used in each extension. Producer threads take each constructing schedule from previous table, extend the schedule and put the extended schedule set into buffer queue. The start time of each actor can be obtained via max-plus algebra [3] by simply calculating $max(maxT, maxP)$, where $maxT$ is the max produced time of tokens that actors need to consume and $maxP$ is the max end time of the last actor on each processor. Taking the extension between $mapTable2$ and $mapTable3$ for example, one producer thread may take schedule $sch2\_1$ in $mapTable2$, and extend it to two schedules by allocating a third firing of actor a on $p1$ and $p2$, respectively. This producer thread puts the two schedules into buffer. Then one consumer thread takes these two schedules and put them into $mapTable3$ which are $sch3\_1$ and $sch3\_2$, respectively.

Algorithm 1 Judge

```
Input: two compared schedule A,B
Output: judge result of A and B
1: if A $\preceq$ B then
2:     return -1 // A is not worse than B
3: else
4:     if B $\preceq$ A then
5:         return 1 // A is worse than B
6:     else
7:         return 0 // can't judge
8:     end if
9: end if
```

Consumer threads take schedules from buffer queues and insert them into the next map tables. For pruning the state space, we compare the new schedule with those in the map table to decide whether to insert it. By judging schedule B in buffer queue with each schedule A with same key in that next map table via Algorithm 1, we insert B when we can't judge, discard it when A is not worse , insert B and remove A when A is worse. Taking the extension between $mapTable1$ and $mapTable2$ for example, $sch2\_2$ is extended by $sch1\_1$ through allocating a firing of a on processor $p2$. We discard $sch2\_3$ because it's the same as $sch2\_2$.

Let $sch[p]$ be the end time of the last firing on processor p and $occT[p]$ be the total occupied time of processor p. Schedule A **dominate($\preceq$)** schedule B is defined as following.

1. $A.sch[p] \leq B.sch[p]$,
2. $A.occT[p] \leq B.occT[p]$, and
3. the start time of each next enable actor of A is earlier than that of B.

Proposition: Method above is an exact pruning policy. We can always find a schedule extended by A which is not worse than that of B if $A \preceq B$.

Proof: According to condition 1 and 3, for any f-schedule B' extended by the constructing schedule B, we can move the firings after B to extend A. The procedure is illustrated in Fig. 3 with shadow boxes. The resulting f-schedule A' has the same thr as B'. According to condition 2, ec of A is apparently not worse than B. So it's proved.

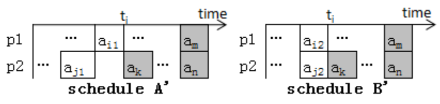


Fig. 3. dominate ($\preceq$) illustration

The state space of scheduling an SDFG can be very large. To further prune the state space, approximate methods can be obtained by replacing the definition of dominate($\preceq$) but may lose accuracy at different degree. The first approximate method we proposed , named *appr*1, is obtained by removing the third condition in the accurate dominate($\preceq$) definition. The second approximate method, named *appr*2, is less accurate than *appr*1. Its dominate($\preceq$) definition is defined as: both of the temporary throughput and energy consumption of constructing schedule A are not greater than that of B.

4 Experiments

We have implemented our algorithms and tested them on two sets of system models on a 2.9GHz server with 32 logical cores, 24M Cache and 384GB RAM. The platforms in system models we considered are 2 processors with different type and 4 processors with two processors per type.

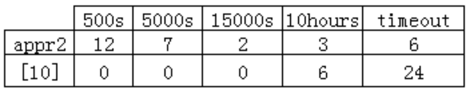
Table 1. Experimental results for MPEG-4 Decoder

info		P5	P10	P30	P5	P10	P30
f	$\#P$	model checking [10]			parallel(exact)		
1	2	2/0.1[a]	2/0.2	3/8.1	2/0.3	2/0.3	3/0.5
	4	1/17.8	1/1221	0/N[b]	1/0.4	1/5.5	0/N
2	2	2/1.2	3/8.3	2/235.5	2/0.4	3/0.6	2/4.3
	4	1/1902	0/N	0/N	1/55.6	2/28479	0/N
		parallel(appr1)			parallel(appr2)		
1	2	2/0.2	2/0.3	3/0.5	2/0.3	2/0.3	3/0.5
	4	1/0.4	1/2.1	2/6978	1/0.3	1/0.3	2/0.5
2	2	2/0.4	3/0.5	2/3.1	2/0.3	3/0.5	2/2.5
	4	1/3.9	2/270	0/N	1/0.3	2/0.5	1/2.6

[a] number of Pareto points/execution time(s).
[b] not finished after 10 hours or running out of memory.

The first case is an MPEG-4 decoder [8] with different parameters. We consider three scenarios, P5, P10 and P30. For each scenario Px, the sum of elements of their q is $3 + 2x$, which means the problem scale grows when x is larger. We compare the model checking methods in [10] with our parallel methods. The results are shown in Table 1. Each cell is filled with the number of Pareto points the corresponding method returns and its execution time in seconds. For small scale problems, model checking method performs as well as the parallel method, while it takes a lot of time or even can't work when problem scale grows. The results show that even our approximate methods hit all the Pareto points while the execution time is much less than the model checking methods for large scale problems.

The second set of case includes some large SDFGs. It is mainly used to show the scalability of our methods. We randomly generate 30 SDFGs using SDF3 tool



	500s	5000s	15000s	10hours	timeout
appr2	12	7	2	3	6
[10]	0	0	0	6	24

Fig. 4. Experimental results for large models

(http://www.es.ele.tue.nl/sdf3) with the sum of elements of their q nearly 1000. The system models we considered are these 30 SDFGs with a platform with two processors. The experimental results illustrated in Fig. 4 present the execution time of the model checking method and our parallel method *appr*2. The number in cell indicates the number of cases solved within these duration. Method *appr*2 can solve 7 cases in 500∼5000 seconds, for example. The results show method *appr*2 can solve 24 of the 30 cases while model checking method solves only 6 cases.

5 Conclusion

In this paper, we have presented a parallel framework for scheduling SDFGs on heterogenous multiprocessor platforms considering the optimization of both throughput and energy consumption. An exact method can be used to obtain all exact Pareto-optimal schedules and two approximate methods have provided a trade-off between accuracy and execution time. Our experimental results show that the execution time of our algorithm is much less than the existing methods for large models while hits all Pareto points for the MPEG-4 decoder case. We will conduct more comparative studies, comparing our methods with other heuristics like list scheduling, in the future.

References

1. Ascia, G., Catania, V., Palesi, M.: Multi-objective mapping for mesh-based noc architectures. pp. 182–187. IEEE (2004)
2. Chen, G., Li, F., Son, S., Kandemir, M.: Application mapping for chip multiprocessors. pp. 620–625. IEEE (2008)
3. Heidergott, B., Olsder, G.J., Woude, J.v.d.: Max Plus at Work: Modeling and Analysis of Synchronized Systems. Princeton University Press (2005)
4. Hu, J., Marculescu, R.: Energy- and performance-aware mapping for regular noc architectures. Computer-Aided Design of Integrated Circuits and Systems 4(24), 551–562 (2005)
5. Lee, E., Messerschmitt, D.: Static scheduling of synchronous data flow programs for digital signal processing. IEEE Trans. Comput 36(1), 24–35 (1987)
6. Murali, S., Coenen, M., Radulescu, A., Goossens, K., Micheli, G.D.: A methodology for mapping multiple use-cases onto networks on chips. In: DATE. pp. 118–123. IEEE (2006)
7. Singh, A.K., Shafique, M., Kumar, A., Henkel, J.: Mapping on multi/many-core systems: Survey of current and emerging trends. In: Proc. of the 50th Ann. Design Automation Conf. (DAC). pp. 1–10 (2013)
8. Theelen, B., Katoen, J.P., Wu, H.: Model checking of scenario-aware dataflow with CADP. In: Proceedings of the Conference on Design, Automation and Test in Europe. pp. 653–658 (2012)
9. Wu, D., Al-Hashimi, B., Else, P.: Scheduling and mapping of conditional task graph for the synthesis of low power embedded systems. Computers and Digital Techniques 150(5), 262–273 (2003)
10. Zhu, X.Y., Yan, R., Gu, Y.L., Zhang, J., Zhang, W., Zhang, G.: Static optimal scheduling for synchronous data flow graphs with model checking. In: Bjørner, N., de Boer, F. (eds.) FM 2015: Formal Methods. LNCS, vol. 9109, pp. 551–569. Springer International Publishing (2015)