

Extending Kahn Process Networks for Timing and Energy Efficient Scheduling

Margarete Sackmann, Dirk Janssens,
University of Antwerp, Middelheimlaan 1,
2020 Antwerp, Belgium

{margarete.sackmann, dirk.janssens}@ua.ac.be

Narasinga Rao Miniskar, Roel Wuyts
IMEC, Kapeldreef 75
3001 Leuven, Belgium

{miniskar, roel.wuyts}@imec.be

Abstract—Embedded systems with multiple cores allow the parallelization of software applications to minimize execution time and energy requirements. In order to fully exploit the parallelization possibilities, applications should be represented by suitable models. We suggest an extension of Kahn Process Networks that is inspired by the scheduling performance of the Gray Box model of Ma et al. [1]. Timing and energy information for the considered hardware platform is added to the model along with real time constraints. Static scheduling that considers interprocessor communication is applied for subgraphs with dataflow behavior.

I. INTRODUCTION

To efficiently schedule computation and memory intensive programs such as multimedia and telecommunications applications on Multi-Processor System-on-Chip (MPSoC) platforms, powerful models for these applications are needed. These abstractions are needed to ensure the portability onto different hardware platforms. The Gray Box model of Ma et al. [1] is a graph-based model that allows parallelization and thus an efficient mapping of software applications to heterogeneous multiprocessor platforms. The model is focused on limiting energy consumption and execution time of the processors. On an embedded system there are however also other components, namely memory and the network connecting the components. Since they also cause time and energy costs, a model that can express data exchange is desirable. In this paper, we explain the scheduling technique of the Gray Box model and suggest a way to use the timing and energy properties of the Gray Box model to extend Kahn Process Networks.

II. THE GRAY BOX MODEL

The Gray Box model, as described in [1], was developed for scheduling real-time processes on embedded systems with multiple processors. The model has two levels: the Thread Node (TN) and the Thread Frame (TF) level, see Figure 1. On the higher TF level, several Thread Frames exist that can be executed in parallel, unless directed edges indicating data or control flow dependencies require in-order execution.

Each TF consists of Thread Nodes that can again be ordered by data or control flow edges. Each Thread Node stands for a part of a program. All the operations in a Thread Node are executed on the same processor. For each invocation of a Thread Frame, every TN that is contained is executed exactly once.

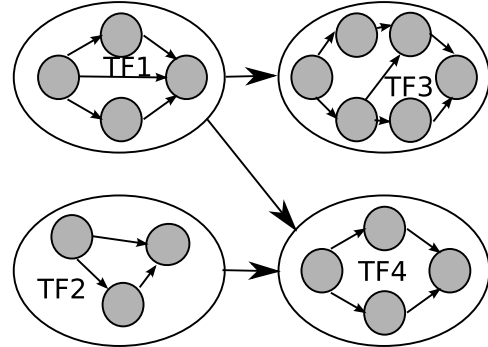


Fig. 1. Gray Box model

This two-level hierarchy is utilized to split the scheduling efforts into design-time and run-time scheduling. At design time, each TF is considered separately. Different ways of assigning the TNs to the available processors are explored, taking timing and precedence constraints into account. Only those schedules that are Pareto-optimal on a multidimensional trade-off curve are kept. At run-time the schedules of the active Thread Frames are combined to form a global schedule.

Formally, Thread Nodes and Thread Frames are defined as follows [1]:

Definition 1 (Thread Node). A Thread Node N is a maximally sized set of connected operations. A Thread Node has a minimum $\delta(N)$ and maximum execution time $\Delta(N)$. $lat(N) = [\delta(N), \Delta(N)]$ is the latency of the Thread Node.

[1] gives instructions on how to identify Thread Nodes, such as an execution time of no more than $100\mu s$ or the fact that all TNs should have roughly the same size. Although the computation within Thread Nodes can be described by Control and Data Flow Graphs, Thread Nodes are regarded as black boxes, where only execution times and energy consumption are known.

Definition 2 (Thread Frame). A Thread Frame is a maximal piece of functionality which has deterministic behavior with a single sequence of control that can be run without rescheduling from the run-time scheduler.

As the contained TNs, their attributes and dependencies

within a Thread Frame are visible, the Thread Frame can be seen as a white box.

III. STRENGTHS AND SHORTCOMINGS OF THE GRAY BOX MODEL

The Gray Box model is specialized for scheduling parallelizable applications on multiprocessor platforms. These software applications can have very strict constraints regarding their execution times. Video decoders for example, require that the frames appear at a certain rate, such as 25 frames per second. In such cases, the timing constraints that can be included on edges in the Gray Box model are extremely useful since they ensure that only schedules that are fast enough will be chosen.

Annotations specify for each Thread Node how much time and energy will be used if that piece of code is executed on the different processors. Since embedded systems often operate with batteries, it is important to keep the energy consumption low. This is considered in the Gray Box model, where schedules can be chosen that are energy efficient.

The two-staged scheduling saves a lot of run time overhead. Since not only one schedule is known for each Thread Frame at run-time, it is possible to combine the schedules for all active TFs without violating timing constraints and at the same time minimizing energy consumption.

The adaption of the Gray Box model to energy efficient scheduling for multiprocessor hardware also brings some disadvantages. The Gray Box model does not have any explicit execution semantics, although for the Thread Frames the token flow semantics of the MTG model described in [2] from which the Gray Box model was derived could be used. This also means that pipelining is not possible and many schedules will be overlooked. Consider a video decoding application that consists of three steps, as shown in Figure 2. Assume that frames A, B, C, \dots should be decoded in that order. Once frame A has passed the first step, frame B can be immediately processed by step 1.

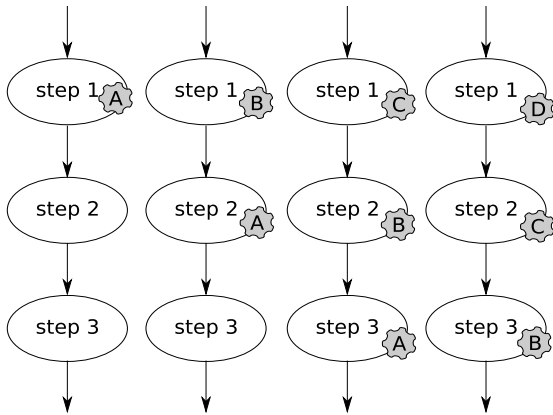


Fig. 2. A model which uses pipelining

Gray Box models depend on the hardware that they are supposed to run on. Consider a Gray Box model being transferred from one hardware platform to a new one that

is ten times as fast. The TNs will almost certainly be too fine grained and the model will have to be reconstructed. A hierarchical model, where processes can be combined to form new processes would be better for scalability.

Some applications running on embedded systems like video or 3D decoders are very data intensive. Consider the example in Figure 3 where four processes $A-D$ are mapped onto a two-processor platform. The table on the right specifies execution costs of the functions on the two processors. The numbers on the edges between two processes define the communication cost if the processes are mapped to different processors. A schedule that considers only execution times maps A, C and D to $P1$ and B to $P2$, causing a communication cost of 25 and overall cost 31. When the communication cost is taken into account, processes A, B and D are mapped on $P2$ while C is mapped on $P1$ causing a total cost of 17. In the Gray Box model it is not possible to consider schedules with interprocessor communication.

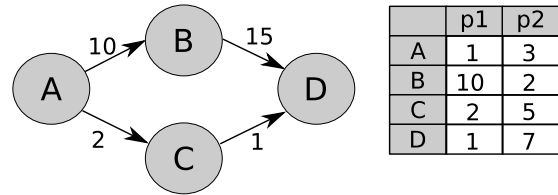


Fig. 3. Communication costs

IV. EXTENDING KAHN PROCESS NETWORKS WITH GRAY BOX MODEL FEATURES

As the Gray Box model is not flexible when considering different hardware platform or communication intensive programs but has a very good scheduling technique that takes energy consumption into account and keeps the runtime overhead low, we propose to use a different model that is extended with features of the Gray Box model.

Kahn Process Networks (KPN) [3] are graph-based models for representing parallel programs. For an example see Figure 4. The nodes of the graph stand for concurrent processes that communicate via unidirectional FIFO channels, represented by arrows. The information on those channels is represented by tokens that can be fed into the channel on one end and are destroyed when they are read. The channels have infinite size so that new tokens can always be added. If a process wants to read data from an empty channel it is blocked until enough tokens are available.

Definition 3 (Functional Process). Let S_1, S_2, \dots, S_p and T_1, \dots, T_q be sets of sequences over the sets D_1, \dots, D_p and D'_1, \dots, D'_q . A *functional process* $F : S_1 \times \dots \times S_p \rightarrow T_1 \times \dots \times T_q$ is a continuous function from a set of input sequences to a set of output sequences.

Definition 4 (Process Network). A *process network* P is a directed graph whose nodes are functional processes and whose edges represent sequences.

Like Petri Nets, Kahn Process Networks are a concurrent model of computation where functions operate on tokens. In KPNs these tokens contain data and the functions operate on this data. The data is put on the channels in a FIFO manner, whereas in Petri Nets, the tokens in places are not ordered and only those in colored Petri Nets have values. Each FIFO queue has exactly one producer and one consumer so that a KPN is deterministic. A process in a KPN may consume and produce a varying number of tokens in each execution.

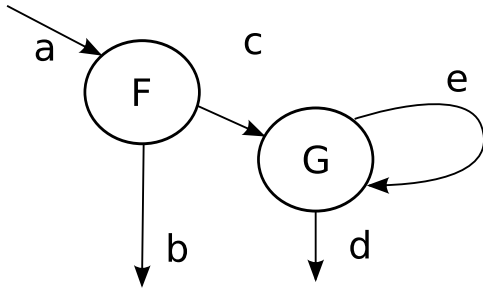


Fig. 4. Kahn Process Network

The semantics of Kahn Process Networks are described by a fixpoint equation that relates the input streams of the network to the output streams. Kahn [3] and Lee and Parks [4] give detailed instructions.

A schedule for a KPN has to decide three things according to Lee and Ha [5]: on which processor each process is executed, in which order the processes are executed and at what time each process starts execution. Scheduling algorithms can be classified by how many of these decisions can be taken before run time.

We suggest to use constraints like in the Gray Box model for the processes and edges. The constraints on the processes can specify the maximally allowed execution time and energy consumption, those on the edges the maximum communication time (minimum delay to start the next process). Control edges can be used to ensure precedence constraints and to define the maximum execution time of a set of processes.

Processes can be annotated with timing and energy consumption for different processor types similar to the Thread Nodes in the Gray Box model. The schedules have to satisfy all timing constraints (on the edges as well as on the processes) and take communication costs into account.

Subgraphs that consist of processes with constant token consumption and production rates should be isolated to find possible schedules for them before run-time, as is done for Thread Frames in the Gray Box model. Heuristics have been proposed for example in [6] to find fast schedules that minimize communication costs for these subgraphs. Rather than finding the subgraph schedules with the lowest execution time, we suggest to start with simple heuristics and successively use more elaborate heuristics until a schedule is found that meets the constraints on the processes and edges of the subgraph. This schedule is then optimized to reduce energy consumption and communication costs, for example by reducing the number

of processors used. Each subgraph can then be regarded as a process at a higher hierarchical level and at run time scheduling methods for Kahn Process Networks such as [7] or [8] are applied to schedule the complete network, similar to the run-time scheduling of Gray Box models.

V. CONCLUSION

Using Kahn Process Networks as a basis, we suggested a model for software applications that can be used for scheduling these applications on multiprocessor platforms in a timing and energy efficient way. Kahn Process Networks can be composed hierarchically, allow pipelining and recursion and can represent communication within an application [6]. We added timing constraints and information on execution times and energy requirements for specific hardware platforms as seen in the Gray Box model. Also from the Gray Box model is the approach to isolate subgraphs with simple, deterministic behavior and schedule them statically. We propose to use heuristics to find schedules that satisfy the timing constraints and then optimize these schedules for energy consumption and communication costs.

ACKNOWLEDGEMENT

This work was supported by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT). This work is part of the project for Optimization of MPSoC Middleware for Event-driven Applications (OptIMMA).

REFERENCES

- [1] Z. Ma, P. Marchal, D. P. Scarpazza, P. Yang, C. Wong, J. I. Gomez, S. Himpe, C. Ykman-Couvreux, and F. Catthoor, *Systematic Methodology for Real-Time Cost-Effective Mapping of Dynamic Concurrent Task-Based Systems on Heterogeneous Platforms*. Springer Verlag, 2007.
- [2] F. Thoen and F. Catthoor, *Modeling, Verification and Exploration of Task-Level Concurrency of Real-Time Embedded Systems*. Kluwer Academic Publishers, 2000.
- [3] G. Kahn, "The semantics of a simple language for parallel programming," in *Information Processing*, J. L. Rosenfeld, Ed., 1974, pp. 471–475.
- [4] E. A. Lee and T. M. Parks, "Dataflow process networks," in *Proceedings of the IEEE*, 1995.
- [5] E. A. Lee and S. Ha, "Scheduling strategies for multiprocessor real-time dsp," in *Globecom*, 1989.
- [6] G. C. Sih, "Multiprocessor scheduling to account for interprocessor communication," Ph.D. dissertation, University of California, Berkeley, 1991.
- [7] J. T. Buck, "Scheduling dynamic dataflow graphs with bounded memory using the token flow model," Ph.D. dissertation, University of California, Berkeley, 1993.
- [8] T. Basten and J. Hoogerbrugge, "Efficient execution of process networks," in *Communicating Process Architectures*, M. M. Alan Chalmers and H. Muller, Eds., 2001.