Resource-aware task graph scheduling using ILP on multi-core

Benjamin Rouxel[*,1], Isabelle Puaut[*,1],
Steven Derrien[*,1]

[*] *Irisa, Université Rennes 1, Campus de Beaulieu, 35000 Rennes, France*

ABSTRACT

Multi-core usage have increased in real-time embedded system. Despite of the different existing multi-core architecture, there is a real need for mapping and scheduling applications on those architecture. Many techniques already exist to achieve it but all of them take the worst-case latency when dealing with shared resource. We propose here to study the impact of synchronisation on the global WCET. By adding synchronisation on tasks that use a shared resources we are able to decrease the effect of contention on shared resources.

KEYWORDS: Real-time embedded system, tasks' graph mapping/scheduling

1 Problem statement

For several years multi-core usage have increased for embedded platforms. Many different architectures showed up some based on buses some other on Network on Chip (NoC). Despite of their differences, multi-core architectures have introduced new interferences conflicts in the estimation of the Worst Case Execution Time (WCET). I/O ports and shared variables are not anymore the only shared resources in the architecture. Now buses and NoCs must also be considered as such shared resources requiring a contention analysis in order to compute a safe WCET [FAFQM+14].

As dealing with shared resources is not a new challenge there exists plenty of arbitration policies that can be used either in hardware either in software. For most of the arbitration policies, there exists a computation to estimate the worst-case latencies. However all arbiters do not offer the same guaranty regarding timing constraints and most of the previous work are using Time Division Multiplexing Arbiter (TDMA) regarding of the poor resource utilisation [KHMF13]. The performance gain brought by the multi-core architecture is then mitigated on the WCET side due to shared resource.

When concurrent tasks running on different cores try to access a shared resource the access order is unpredictable. Thus when computing the WCET of such tasks the worst-case concurrency must be added to the WCET of all tasks. This pessimism can be reduced by using synchronisations in order to avoid interferences. In this work we will use mutual exclusion.

[1]E-mail: {firstname}.{lastname}@irisa.fr

Before running application on multi-core, there must be a mapping/scheduling phase to decide what code will run on what core (mapping) and in what order (scheduling) [DB11]. We consider here the mapping/scheduling step to be statically computed with an exact method based on Integer Linear Programming (ILP). Despite of the scaling problem induced by exact method which is NP-Hard[CJGJ96], when the solver finds a schedule it is an optimal one and it can be used as a reference for further heuristics.

In this work, we want to compare the result of the mapping/scheduling process with and without synchronisation. This absence of synchronisation is illustrated by overlapping the execution of tasks. Thus we have two mapping/scheduling methods, first one mutually excludes concurrent tasks (no need for specific software/hardware addition) that accesses the bus, and second one allows tasks to overlap their execution. Obviously the latter one will also integrate the worst-case interference bound while the former will not.

2 Task model

In this study we target streaming applications[GTA06]. This type of applications can be represented with a Directed Acyclic Graph (DAG) called a task graph. A task graph is defined by a set of tasks and a set of edges linking causally dependent tasks. Two tasks are causally dependent if one needs the other one finishes before starting.

A task is defined by its release time and its WCET. An edge is defined a source task, a target task and the amount of data exchanged between those tasks. An edge is present in the graph when a target task needs to receive data from a source task.

To exhibit the part of tasks that use the communication medium, we surround each task witha receiving (placed before) and a sending task (placed after) as in [TPGM14].

For each task we need to estimate their WCET in isolation. Such information relies on the target processor.

3 Architecture and Communication model

We target processor architecture with scratchpad memory (SPM) where the communications between cores are done from the sender SPM to the recipient SPM through the communication medium. This architecture is based on a simple statically predictable processor called Patmos[SBH+15] from T-Crest project[2].

As a simple case, we choose to connect cores with a bus as the communication medium with Time Division Multiplexing (TDM) and Round-Robin (FAIR) arbitration policy. For TDM policy, the time is divided in incompressible time slot of fixed duration. Each core has a set of time slot allocated. At the opposite FAIR policy enqueues access requests (one queue per core) and serves them in order with a maximum allowed delay. The major drawback of TDM arises when a task does not use all the allocated time to transmit data, there is a time wastage during which the shared resource is not used.

The worst-case interference latency depends on the amount of data to transmit, the number of processors and the configuration of the arbitration policy. This can be summarised by a $waiting\ time\ +\ time\ to\ send\ data$.

[2]http://www.t-crest.org/

The *waiting time* corresponds to the worst-case waiting time of the shared resource due to the concurrency with other processors. This time is not dependent of the amount of data. For TDM arbitration, the worst-case scenario for a task is when its slot just ends up, and the task has to wait for the next one.

The *time to send data* represents worst-case scenario for sending all the data, for example with TDM, if the amount of data to send do not fit in one time slot, it will also contains some other waiting time between allocated slots.

What we want to expose here is the afore mentioned *waiting time*. Indeed this period is mandatory when task can be in concurrency (they are overlapping), but can be removed when tasks are correctly synchronized (in mutual exclusion).

4 Solving method

To solve the mapping/scheduling problem of a task graph on a multi-core architecture we use an ILP formulation. This exact method is based on a constrained system which then feed an ILP solver. In our experiments we use the CPLEX solver[3]. Our scheduling model is time-triggered clustered non-pre-emptive. This imply we do not have synchronisation cost.

Due to space consideration we will not give the whole constraint equations. But the intuition is to have variables to represent the mapping of a task on a processor, variables to represent the task's ordering, and also some to represent the release time of each tasks. Those variables would then be constrained with the following:

- Unicity constraint: a task is mapped to only one processor
- Conflict constraints: ordering task, a task that is before an other one can not be after this same one
- Causality constraint: ordering task in accordance with the dependencies in the task graph
- Communication task specificities: forcing communication task to be on the same processor as computation one (respecting the order)

In addition, the objective function is to minimize the schedule length. This means minimizing the time at which the last task ends its execution.

5 Conclusion

The concern of this work in progress is mapping/scheduling task graph on multi-core architecture using ILP. We aim to schedule streaming applications. We target multi-core architecture where cores contain a SPM and are connected through a bus. We start to have encouraging results (no presented due to space limitation) by comparing two arbitrations policies for the bus: TDM and FAIR.

The contribution is the comparison of the usage of synchronisation during the mapping/scheduling phase depending on the bus arbitration policy. Indeed we are able to show that for an increasing number of processors, it is best to use mutual exclusion with FAIR.

[3]http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

To go further we are aiming to map and schedule the StreamIT benchmark suite[TKA02] in order to validate our method. We also will improve our ILP formulation to let the the solver decides when overlapping or synchronising couple of tasks, leading to a partial synchronised/overlapped schedule. Also we would like to allow NoC architecture to profit of our method as current many-core are mostly based on this type of communication medium. Finally, due to the complexity of exact method, we will work on heuristics to well scale larger task graphs.

References

[CJGJ96] Edward G Coffman Jr, Michael R Garey, and David S Johnson. Approximation algorithms for bin packing: a survey. In *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Co., 1996.

[DB11] RI Davis and A Burns. A survey of hard real-time scheduling algorithms for multiprocessor systems. *in ACM Computing Surveys*, 2011.

[FAFQM+14] Gabriel Fernandez, Jaume Abella Ferrer, Eduardo Qui nones Moreno, Christine Rochange, Tullio Vardanega, Francisco Javier Cazorla Almeida, et al. Contention in multicore hardware shared resources: Understanding of the state of the art. 2014.

[GTA06] Michael I Gordon, William Thies, and Saman Amarasinghe. Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. In *ACM SIGOPS Operating Systems Review*, volume 40, pages 151–162. ACM, 2006.

[KHMF13] Timon Kelter, Tim Harde, Peter Marwedel, and Heiko Falk. Evaluation of resource arbitration methods for multi-core real-time systems. In *WCET*, pages 1–10, 2013.

[SBH+15] Martin Schoeberl, Florian Brandner, Stefan Hepp, Wolfgang Puffitsch, and Daniel Prokesch. Patmos reference handbook. *Technical University of Denmark, Tech. Rep*, 2015.

[TKA02] William Thies, Michal Karczmarek, and Saman Amarasinghe. Streamit: A language for streaming applications. In *Compiler Construction*, pages 179–196. Springer, 2002.

[TPGM14] Pranav Tendulkar, Peter Poplavko, Ioannis Galanommatis, and Oded Maler. Many-core scheduling of data parallel applications using smt solvers. In *Digital System Design (DSD), 2014 17th Euromicro Conference on*, pages 615–622. IEEE, 2014.