Worst-Case Communication Overhead in a Many-Core based Shared-Memory Model

Amira Dkhil, Stéphane Louise, Christine Rochange

Worst-Case Communication Overhead in a Many-Core based Shared-Memory Model

Amira Dkhil
CEA-LIST, Nano-Innov
PC172, F91191
Gif-sur-Yvette Cedex, France
amira.dkhil@cea.fr

Stéphane Louise
CEA-LIST, Nano-Innov
PC172, F91191
Gif-sur-Yvette Cedex, France
stephane.louise@cea.fr

Christine Rochange
IRIT, Université de Toulouse
118 route de Narbonne. 31062
Toulouse cedex 9, France
rochange@irit.fr

ABSTRACT

With emerging many-core architectures, using on-chip shared memories is an interesting approach because it provides high bandwidth and high throughput data exchange. Such a feature is usually implemented as a multi-bus multi-banked memory. Since predicting timing behavior is key to efficient design and verification of embedded real-time systems, the question that arises is how to evaluate the access time for one memory access of a given task while others may concurrently access the same memory-bank at the same time. In this paper, we give the answers for a subset of streaming applications modeled like CSDF Model of Computation and implemented in Kalray's MPPA chip.

Keywords

Access time, Shared memory, Multi-core, CSDF, MPPA chip

1. INTRODUCTION

Predicting timing behavior is key to efficient design and verification of embedded real-time systems. For embedded hardware platforms, Multiprocessor Systems-on-Chip (MPSoCs) provide a good balance between cost, power efficiency, and flexibility. Multi-core systems are known to be especially difficult regarding Worst-Case Execution Times (WCETs), but the new way to program these many-cores is different from the way embedded systems used to be programmed with micro-controllers: Dataflow Models of Computation (MOCs) are gaining in momentum because some subsets of the dataflow based model of computation possess good properties concerning parallelism management [4]. It has been shown recently that over 90% of streaming applications can be modeled as acyclic SDF graphs [2]. Cyclo-Static Dataflow (CSDF), the model we consider, is a generalization of SDF [8] in which consumption and production rates take the form of periodic sequences. CSDF is more versatile because it also supports algorithms with a cyclically changing, but predefined, behavior. It can provably run without locks and for well-formed applications (detectable at compilation time) in finite and statically-known memory.

Typically, these applications are partitioned into tasks that communicate over channels (i.e. FIFO buffer) together forming a Dataflow graph. In order to allow maximum flexibility at the lowest cost, tasks share storage, computation and communication resources. This leads to uncertainty about resource management which can make the system decomposable. The temporal behavior of each task becomes dependent on other tasks and cannot be analyzed in isolation. As system runs without locks, what is required to know to calculate worst-case latencies, in addition to stand-alone WCETs of individual tasks, is the communication overhead induced by the interference when several processors want to access nearly simultaneously to the same bank of shared memory. We will show that for a class of periodic scheduling schemes called implicit-deadline periodic schedule, it is possible to compute worst-case communication overhead in shared memory clusters of Kalray's MPPA chip [6] for a subset of usual stream programs with the task resulting from the compilation of the Sigma-C associated dataflow language [7]. Bamakhrama and Stefanov [2] proved that Implicit-Deadline Periodic (IDP) scheduling approach gives the maximum achievable processor utilization and throughput for large set of dataflow graphs, called matched I/O rates graphs [2]. These graphs represent more than 80% of streaming applications [2]. Self-timed schedule (STS), also known as as-soon-as possible schedule, was considered the most appropriate for streaming applications modeled as dataflow graphs [2, 3]. Moreira and Bekooij [3] established that it is possible to guarantee strictly periodic behavior of tasks within self-timed implementation. They have also provided the maximum latency for applications with periodic, sporadic and bursty sources. In [1], authors present a complete framework for computing the periodic task parameters using an estimation of worst-case execution Time. They assume that each write or read has constant execution time which is often not true.

Our approach is similar to [1, 2] in using the periodic task model which allows applying a variety of proven hard-real-time scheduling algorithms for multiprocessors. However, it is different in exploring system level in order to get upper bounds on the communication overhead as close as to the real values. Ongoing approaches focus on performance analysis either on task or system level. Especially memory accesses cannot be accurately captured on a single level alone: considering both perspectives lead to overly pessimistic estimations.

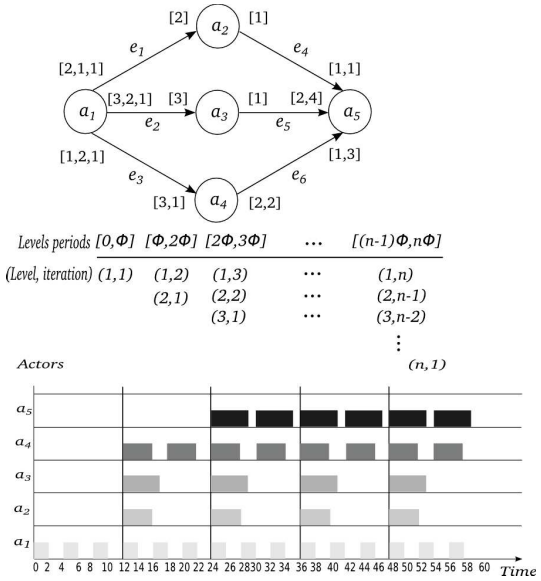

Figure 1: IDP schedule for CSDF graph

1.1 Motivating Example

We show the possible influence of parallel execution of actors, under static implicit-deadline periodic (IDP) schedule, on accesses to memory by means of an example. Figure 1 illustrates a CSDF graph consisting of five actors and six communication channels. Under IDP schedule, actors are executed in levels. Actors are assigned to levels according to the dependency and the minimum number of tokens that should be present on communication channels before firing. $A_j(n)$ is the set of actors in j level executing their n^{th} iteration. Actors in level j start execution at $t = (j-1) \times \phi$. ϕ is the global level period [1]. From the example depicted in Figure 1, at $t = 24$, actor a_5 of level 3 starts execution of his 1^{st} iteration, actor a_1 of level 1 executes his 3^{rd} iteration, actors a_2, a_3, $and\, a_4$ of level 2 execute their 2^{nd} iteration. The periodic start time $t = (j-1) \times \phi$ guarantees that actors in a given level will have enough data (i.e. from their predecessors) to start. From [2], authors proved that such schedule executes with bounded memory buffers. Thus, all actors of level j can start their execution simultaneously at $t = (j-1) \times \phi$ and surely finish at $t = j \times \phi$. Well, it is simple to observe, from Figure 1, that actors can read or write memory at the same time. In this example, actors execute on five processors. A processor can request access to the shared memory without restriction but not without penalties, since there is an additional and variable arbitration cost. It would be trivial to assume that this cost is the same for all accesses to memory because this will induce a very pessimistic and maybe unreliable result. For these reasons, we must look more closely at the access modes of shared resources so that it will be possible to derive a fairly accurate estimation.

1.2 Paper Contributions

Given a streaming application modeled as an acyclic CSDF graph with periodic input streams, determine the maximum number of overlapping reads and writes when executing actors as implicit-deadline periodic tasks. For each level in the CSDF graph, define the worst-case communication overhead induced by the interference when several processors want to access simultaneously to the same bank of shared memory. The communication costs are defined as follows: 1) Arbitration cost: the time needed to arbitrate shared communication resources at run-time, 2) Synchronization cost: the time needed to check if all the necessary data is available for the actor and 3) Transfer delay: The mean-time needed to transfer input and output tokens from and to the private memory of processing elements.

The remainder of this paper is organized as follows: Section 2 introduces the CSDF model, the system model and the IDP schedule. Section 3 defines theoretical results. Finally, in section 4, we present our case study and then we conclude.

2. BACKGROUND

2.1 Cyclo-Static Data-Flow (CSDF)

We use Cyclo-Static Dataflow [9] to model real-time streaming applications. It is a directed graph $G = (A, E)$, where A is a set of computation actors and E is a set of communication channels. Data is transported in discrete chunks, called tokens, via communication channels implemented as First-In First-Out (FIFO) queues. An actor is enabled by the availability of enough tokens on each of its incoming edges. This is done by means of synhronization mechanisms like semaphores. An enabled actor can fire and consume/produce fom/to each of its input/output edges a number of tokens. Actor firings are free from side effects. Each actor $a_i \in A$ is viewed as executing through a periodically-repeating sequence of functions $[f_i(1), f_i(2), ..., f_i(\tau_i)]$ of length $\tau_i \in \mathbb{N}^\star$. P and C are the sets of production and consumption rates. For example, the j^{th} firing of a_i is enabled if there is at least $[c_i^{e_i}(((j-1) \mod \tau_i)+1)]$ on its input channel e_i, when fired, it executes the code of function $f_i(((j-1) \mod \tau_i)+1)$ and produces $[p_i^{e_o}(((j-1) \mod \tau_i)+1)]$ tokens on its output channel e_o.

2.2 System Model

Late last year, two new architectures have emerged: the SThorm chip from STMicroelectronics (i.e. 64 cores) and MPPA chip from Kalray [6] (i.e. 256 cores). These chips rely on a clustered architecture that allows clusters of processors to share a particular level of the memory hierarchy and this has the potential to reduce the average memory access time of parallel applications [5]. A single MPPA cluster consists of 16 user processors, a controller processor, and a shared banked memory. It also comprises two DMA engines (one in and one out) to exchange data with external parts of the cluster through the NOC interface, but this is out of the scope of this paper. Each processor has private L1 cache and communicates with other processors through a shared banked memory (SRAM) of 2MB. The banked memory is implemented as a multi-bus approach [6]: it provides the same functionality as a full crossbar with lower impact on surface occupation or power consumption. Each memory-bank has a private controller which manages the requests sent from each processor in the cluster using a FIFO (first-in, first-out) equivalent queuing strategy: this will give rise to extra penalties. The shared memory is a Static RAM (SRAM), so it is quite feasible to derive some access time: some time is spent in sending a request to the controller,

and once the request is satisfied, back to the processor, this time is noted t_0. t_0 is constant because there is no memory coherence protocol. But Since we have overlapped accesses, the overall access time is:

$$t = t_0 + (\pi_j - 1) \times t_c \quad (1)$$

t_c is the time needed from the controller to access memory and it costs one RAM cycle [6]. $(\pi_j - 1)$ models the order of processor requests in the FIFO of the memory-bank controller. π_j is the number of processors executing actors of level j.

2.3 Implicit-Deadline Periodic Schedule

Under IPD static schedule, processors execute a task set $A = [A_1, A_2, ..., A_n]$ of n periodic tasks. In this paper, we consider that a task cannot be preempted during execution. A periodic task $A_j \in A$ is defined as a 4-tuple $A_j = (S_j, \omega_j, \lambda_j, D_j)$ where S_j is the start time, ω_j is the worst-case execution time, λ_j is the task period and D_j is the relative deadline of A_j. The k^{th} invocation of task A_j is at time instants $t = S_j + k \times \lambda_j, \forall k \in \mathbb{N}$. A_j executes for ω_j time units and his execution time should not exceed D_j. A task has an implicit deadline if $D_j = \lambda_j$, it follows that A_j has to terminate before time $t = t = S_j + (k+1) \times \lambda_j$. The authors in [2] explain the following definitions in more details: Since actors of CSDF graph G are assigned to levels, we define ϕ as the minimum level period and λ as the minimum actor period. These periods are given by the solution to both equations:

$$\phi = q_1\lambda_1 = q_2\lambda_2 = ... = q_n\lambda_n \quad (2)$$

and

$$\vec{\lambda} - \vec{\omega} \geq \vec{0} \quad (3)$$

where $\vec{q} = [q_1, q_2, ..., q_n]$ is the repetition vector of G and $q_j \succ 0$ represents the number of invocations of an actor a_j in a valid schedule of G. G is consistent if there exists a repetition vector: When each actor is fired the number of times specified by $\vec{q}$, the total number of tokens produced on each arc is equal to the total number of tokens consumed.

3. WORST-CASE OVERHEAD

3.1 Assumptions and Definitions

A graph G refers to an acyclic consistent CSDF graph. A consistent graph can be executed with bounded memory buffers and no deadlock. So, we only consider consistent and deadlock free CSDFs. Consistency concerns the correspondence between production and consumption rates [11]. For our analysis, we assume the following hypothesis:

H1- Basic access time of every actor to shared memory conforms to (1). It is defined as the total time needed to access memory depending on the order of access to FIFO controller.

This order can be determined if we have exact knowledge of each access requesting time which is not feasible in practice. Under static or dynamic schedules, the order of accesses to memory cannot be determined, even for fully static schedule where we assume a very tight estimation of worst-case execution time of actors. In [12], Khandalia et al. explored the problem of imposing an ordering of interprocessor communication operations in statically scheduled multiprocessors. Their method is based on finding a linear ordering of communication actors at compile time which could minimize synchronization and arbitration costs, but this would be at the expense of some run-time flexibility. In this paper, we do not impose any constraints on communication operations.

Definition1: For a graph G under IDP schedule, the worst-case overhead O_j of level j depends on the maximum number of accesses to memory $m_{i,j}$ of actor a_i not on the exact time when a processor requests an access:
$O_j = f(m_{i,j}), \forall a_i \in A_j$. A_j is the set of actors of level j.

H2- Reading or writing tokens from/to the memory could be done at any time. This assumption was derived from simulation results: during execution, the processor may require read access when it needs some data and write access when it finishes a part of the execution.

This last assumption does not affect the considered size of shared buffers with IDP periodic schedule because reading and writing in the same shared buffer cannot be done in the same time as the execution is periodic and ordered in levels.

H4- We assume that we have reasonably tight estimates of actors computation time. Computation time is the time needed for computation operations. These estimates can be obtained by several different mechanisms like those described in [10].

H5- In [3], synchronization checks are done whenever processors communicate: the sending processor ascertains that the buffer it is writing to is not full, and the receiver ascertains that the buffer it is reading from is not empty. For IDP schedule, the synchronization cost is equal to zero, because periodic behavior guarantees that an actor a_i will finish execution before deadline D_i.

3.2 Tight overhead under IDP Model

For a simplified problem with only few processors and few concurrent tasks with few accesses to memory, the worst-case communication overhead cannot affect so much the worst-case latency of the application. The non-obvious result is that for such a configuration when the number of processors and accesses to a single-bank memory are very high, the mean access time is deeply impacted by the concurrent accesses. In this section, we introduce an execution scheme to determine an upper bound for overhead.

The CSDF graph is denoted $G_\omega = (A, E, \omega)$, ω is the set of worst-case computation time of actors. Let $S = (G_\omega, \beta, \sigma)$ be the IDP model applied to G_ω. β is the set of levels resulting from scheduling and σ is the number of levels.

Let $\beta = [\beta_1, \beta_2, ..., \beta_\sigma]$ be the set of actors for each level. $\beta_i = [\beta_i^1, \beta_i^2, ..., \beta_i^{\pi_i}]$ of level i, $\forall i \in [1, \sigma]$ is the set of actors for each level in each executing processor. π_i denotes the number of processors executing level i.

$\forall a_j \; such \; that \; a_j \in \beta_i$, it is possible to derive the associated maximum number of accesses to memory (H1). We define $M = [M_1, M_2, ..., M\sigma]$ the set of memory accesses for all levels such that $M_i = [0, m_i^1, m_i^2, ..., m_i^{\pi_i}]$ is the set of total number of memory accesses in each processor executing level i. Note that a given processor can have multiple tasks in each level.

M_i is sorted in this order: $\forall i \in [1, \sigma], \quad \forall j \in [1, \pi_i] \quad m_i^1 \leq m_i^2 \leq ... \leq m_i^{\pi_i}$, such that $m_i^{k+1} \neq m_i^k, \forall k \in [1, \pi_i]$. The new dimension of vector M_i is noted α_i. In order to get a tight overhead estimation, we assume that, for the $(\pi_i - 1)$ potential concurrent tasks on a single memory bank, the minimum number of accesses in a given level have the maximum

overhead (H2). Thus we can guarantee safe estimation. The worst-case arbitration overhead of level i is given by:
$O^i_{arb} = 0,\ if\ \pi_i = 1$
$O^i_{arb} = (m^1_i - 0) \times 1 \times t_c,\ if\ \pi_i = 2\,, \alpha_i = 3$
$O^i_{arb} = (m^1_i - 0) \times 2 \times t_c + (m^2_i - m^1_i) \times 1 \times t_c,\ if\ \pi_i = 3\,, \alpha_i = 4$
.
.
.
Where:

$$O^i_{arb} = \sum_{\delta=1}^{\alpha_i - 2} (m_i^{\delta+1} - m_i^{\delta}) \times (\pi_i - \delta) \times t_c \qquad (4)$$

Equation 4 implies that the minimum difference in the number of accesses of processors, for a given level, will get the maximum overhead and so on. This allows us to get a tight estimation of worst-case overhead since accesses will get a variable penalty. Using Equation 4, we can derive the worst-case overhead of level i by adding the transfer delay:

$$O_i = \sum_{\delta=1}^{\alpha_i - 2} (m_i^{\delta+1} - m_i^{\delta}) \times (\pi_i - \delta) \times t_c + \underset{j=1 \longmapsto \pi_i}{max} (m_i^j \times t_0 + \omega_j) \qquad (5)$$

From (5), Equation (3) becomes:
$\vec{\lambda} - \vec{\omega} - \vec{O} \geq \vec{0}$
Thus, we can take into account communication overhead in estimating IDP periods.

4. EXPERIMENTS

We evaluated our proposed temporal analysis scheme on four programs belonging to the StreamIT Benchmarks: Direct Cosine Transform, Biotonic Sort, Audio Beam former and Laplace transformation. As architecture platform, we use the Kalary MPPA multi-cluster multi-core architecture and the associated shared memory arbitration mechanism. This paper does not consider the presence of a 2 two-way instruction and data caches for each processor of the cluster. The execution time analysis is done in three steps. First, the application is executed with a set of input data on the architecture and an execution trace of memory accesses is generated. Second, we apply the scheduling strategy in order to delimit the different phases of execution. Finally, we derive the number of memory accesses and the execution time for each node in the data-flow graph and use these informations to compute the worst-case communication overhead. The simulation time to generate these parameters was in order of minutes, the longest time was spent to derive the memory trace of each node in the application graph, because most of them contains over 60 nodes.

In the case of the MPPA cluster, π is valued between 1 and 16. As a result $(\pi_i - j)$ is bound by 15 cycles, which is the worst-case overhead associated to a given memory access. t_0 costs seven cycles [6]. For the relevant cases, in Figure 2, the worst-case overhead is between 17% and 23% of the overhead derived from simulation results.

5. CONCLUSION

The main contribution of this paper is to propose a safe timing per-cluster access memory model. This is a novel approach proposed in order to estimate the worst-case communication overhead. It was, as far as we are aware, the first

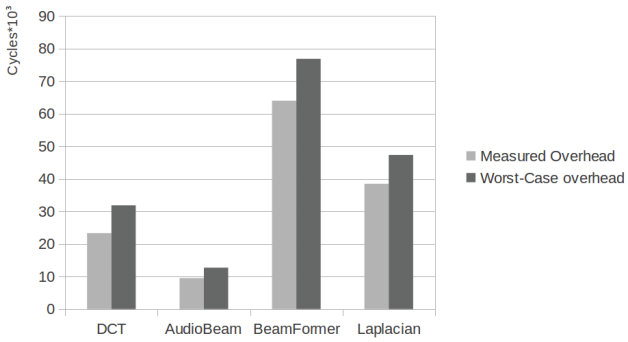

Figure 2: Measured overhead and Worst-case overhead

time that a precise estimation of communication overhead was provided for such an architecture. The evaluations are also compliant with the experimental results. From the case study, we conclude that the timing model is very accurate and significantly improves the precision of worst-case overhead. We will also apply the same methodology for other scheduling strategies. We would like also to determine the communication overhead if accesses are distributed between the different memory banks in the same time.

6. REFERENCES

[1] M. A. Bamakhrama and T. Stefanov, *Managing Latency in Embedded Streaming Applications under Hard-Real-Time Scheduling*. CODES+ISSS, 2012.

[2] M. A. Bamakhrama and T. Stefanov, *Hard-real-time scheduling of data-dependent tasks in embedded streaming applications*, EMSOFT, 2011.

[3] O.M. Moreira and M.J.G. Bekooij, *Self-Timed Scheduling Analysis for Real-Time Applications*, 2007.

[4] S.Sriram and S.S. Bhattacharyya, *Embedded Multi-processors: scheduling and synchronization*. Marcel Dekker, 2009.

[5] A. Erlichson et al. *The Benefits of Clustering in Shared Address Space Multiprocessors: An Applications-Driven Investigation*,1995.

[6] B. D. Dinechin et al., *A distributed run-time environment for the kalray mppa-256 integrated manycore processor*. ICCS Alchemy Workshop, (to be published), 2013.

[7] T. Goubier et al., *ΣC: A programming model and language for embedded manycores*, 2011.

[8] E.Lee and D. Messerschmitt, *Synchronous dataflow*. IEEE Proceedings, 1987.

[9] G. Bilsen et al., *Cyclo-static dataflow*. IEEE trans. Signal Process, Feb. 1996.

[10] R.Wilhelm et al.. The worst-case execution-time problem_overview of methods and survey of tools, 2008.

[11] E. A. Lee, *Consistency in dataflow graphs*, 1991.

[12] M. Khandelia et al., *Contention-Conscious Transaction Ordering in Multiprocessor DSP Systems*, 2006.