Dataflow-based Modeling and Performance Analysis for Online Gesture Recognition

Florian Grützmacher, Benjamin Beichler, Christian Haubelt
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18051 Rostock, Germany
Email: florian.gruetzmacher@uni-rostock.de

Bart Theelen
Embedded Systems Innovation by TNO
PO Box 6235, 5600 HE Eindhoven, Netherlands
Email: bart.theelen@tno.nl

***Abstract*—Cyber-Physical Systems (CPS) are tightly coupled with the environment, and therefore it is important that interactions with the surroundings like Human-Computer-Interactions are performed very responsive. Since CPS are often embedded without traditional input devices, like in medical or automotive contexts, gesture recognition approaches are emerging. As those algorithms are computationally complex especially when implemented on multi-core architectures, design decisions have to be taken carefully in order to meet performance and energy constraints. In this paper, we present a Scenario-Aware Dataflow model to estimate the performance of a template-based hand gesture recognition system based on Dynamic Time Warping (DTW). Our model enables us to estimate the important characteristics like real-time capabilities for online recognition and response time of the system when implemented on a multi-core architecture. Moreover, we introduce an extension to existing SADF performance analysis tools, which enables us to acquire processor utilization from our model. Based on the performance estimations the real-time capability for online recognition was validated for different configurations and verified in our experiments.**

I. Introduction

Gesture recognition is a promising way for Human-Computer Interaction (HCI) especially for Cyber-Physical Systems. As those usually contain a lot of inertial measurement units, sensor-based gesture recognition has become very promising. However, the recognition of complex patterns demands for both, high accurate sensors as well as a computing platform with sufficient computational power. Especially, real-time online gesture recognition is a challenging task.

Multi-core architectures find their way into Cyber-Physical Systems more and more and already provide high computational capacities, which are important for processing dataflow oriented applications like sensor-based gesture recognition. In order to exploit the offered resources and to simplify the implementation process, model based development methods have proven to be effective. One of the main advantages of early system models is the ability to predict system characteristics and therefore the possibility to substantiate early design decisions. Furthermore, sophisticated formal models like dataflow graphs allow the estimation of extra-functional properties like computing performance and energy consumption of the system.

In this paper, we use Scenario-Aware Dataflow (SADF) graphs in order to model different modes of operation depending on the change of sensor signals of a gesture detection system running on multi-core architectures and estimate the expected performance. SADF graphs extend Synchronous Dataflow (SDF) graphs by means of modeling dynamic behavior while still supporting formal analyzability and the ability to effectively map computations onto multi-core systems. We developed an SADF model and a multi-core implementation of a template-based hand gesture recognition system based on Dynamic Time Warping (DTW). The SADF model enables us to predict the real-time capabilities of the online recognition system and response time of the system for different parallelization strategies. Moreover, we are able to estimate the average utilization of the computing platform, which directly influences the power consumption of the system. In our experiments, we fitted and calibrated this system for the Texas Instruments multi-core DSP TMS320C6678 containing 8 Digital Signal processor (DSP) cores.

In comparison to previous work, our contributions are:

1) We present an SADF model, which is configurable in order to represent different mappings onto multi-core architectures.
2) We propose to use scenarios in order to distinguish between low and high fluctuating sensor signals, i.e., sensor data which most probably contain no gesture/significant movement at all, and sensor data which might contain gestures, thus requiring the computation of the recognition algorithm.
3) We propose a novel way to estimate the system utilization by extending the SADF performance analysis tools and compare the performance estimation results with our multi-core DSP implementation.

Our paper is structured as follows. First, related work is discussed. Section III formally introduces Scenario-aware Dataflow (SADF) graphs and Synchronous Dataflow (SDF) graphs. This is followed by the explanation of the proposed parallelization approach for template-based gesture recognition on muti-core architectures. The key contribution of the paper is presented in Section V, where the SADF model of the recognition system is presented in detail. The experiments to validate the prediction of extra functional model properties are described in section VI, their results are presented in section VI-A. In Section VII conclusions are drawn and further research is outlined.

II. Related Work

Studies on different classification algorithms for sensor-based gesture recognition have shown that using template-based approaches like *Dynamic Time Warping* (DTW) show good recognition accuracies [1][2][3]. Based on those studies we focused on template-based gesture recognition and implemented a gesture recognition system based on DTW in our experiments although our approaches can be applied to any template-based gesture recognition task.

In the past, research on exploiting parallelism within the DTW algorithm has been done. Burr et al. could speed up the template matching process by exploiting instruction-level parallelism. However specialized hardware is needed for their approach [4]. Bae and Fairhurst recommend globally shared memory architectures with fast interconnection networks to avoid impacts of inter-core communication to the performance of DTW when exploiting instruction-level parallelism [5]. Yoder and Siegel could achieve ideal speedups using thread-level parallelism for template-based gesture recognition with DTW. However, their approach cannot be applied effectively when having more processing cores than templates [6]. In [7], a configurable approach using thread-level parallelism in template-based gesture recognition has been introduced, which overcomes this problem and enables a compromise between latency and scalability on multi- and many-core systems. An extended dataflow model based on Enable/Invoke Dataflow (EIDF) [8] has also been provided to formally describe this approach. However, this model has no capabilities to capture timing information of the modeled system.

In order to substantiate implementation decisions in a very early design stage, we have to take into account the system performance. Therefore, we need a model from which performance metrics can be calculated for different configurations of our parallelization approach. We used Scenario-Aware Dataflow which has been introduced in [9]. Those models have already been applied for MP3 and MPEG-4 decoders [10][11]. They have also been used for selecting proper *Dynamic Voltage Frequency Scaling* modes for several streaming applications. In contrast, in our paper we use *Scenario-Aware Dataflow* (SADF) graphs to select proper parallelization configurations of the online gesture recognition system regarding real-time requirements and latency when performed on multi-core processors. Moreover, scenarios are used to distinguish between different processing modes which depend on the fluctuation of the sensor signals. Signals with low fluctuations are skipped during gesture detection as no significant movement is contained in the signal. Only sensor signals with high fluctuations are considered for gesture recognition. We also propose a novel way to use SADF performance analysis tools to estimate processor utilization for different parallelization strategies. Resource utilization for SADF has also been addressed in [12] and [13]. In [12], a non-deterministic state machine to capture dynamic applications and their mapping on resources is exploited. Assuming a similar state machine, [13] relies on extending SADF with additional concepts to capture resource usage. Our work relies on the stochastic variant of SADF to capture occurrences of sensor signal fluctuations and proposes an approach to compute processor utilization without extending SADF.

III. Scenario-Aware Dataflow Graphs

The conceptual basis of the presented modeling approach are Synchronous Dataflow (SDF) graphs as described by [14]. An SDF graph $G = (V, E, cons, prod, D)$ consists of a set of Vertices V, a set of edges $E \subseteq V \to V$, token consumption rates $cons : E \to \mathbb{N}$, token production rates $prod : E \to \mathbb{N}$, and a delay function $D : E \to \mathbb{N}_0$. The vertices are so called *actors* communicating data tokens over unbounded channels with FIFO semantics represented by edges, so every channel is annotated with the number $d(e)$ of tokens on it. In SDF graphs the consumption and production rates need to be fixed. An actor $v \in V$ can be fired if $\forall e = (\widetilde{v}, v) \in E : d(e) \geq cons(e)$. If actor v fires, it consumes $cons(e)$ token from each incoming edge $e = (\widetilde{v}, v) \in E$ and produces $prod(e)$ token on each outgoing edge $e = (v, \widetilde{v}) \in E$. The execution of an SDF graph forms fixed repetitive firing sequences caused by the constant production and consumption rates. Every such sequence is called an *iteration* and could be described by a non-trivial repetition vector γ, which describes the number of activations (firings) of every actor to get into a recurring state (count of tokens on the channels).

In [9] the semantics of SDF was extended to Scenario-Aware Dataflow (SADF) with the concept of scenario-depended execution parameters of actors. Therefore every actor can operate in different modes, which define separate token rates and execution times. Additionally new entities within the SADF graphs are defined. New vertices called *detectors* execute discrete-time Markov chains to maintain the scenario transition logic of a subset of actors, which are controlled by this detector. Markov chains consist of a finite state space and probability-based transitions between these states. Therefore the state of the detector is not dependent on the actual values of data token from its input channels. Every controlled actor has a so called *control channel* from a detector, which transports *control tokens*. For every execution of an actor with different modes, a control token is consumed that determines the current operation mode of this actor. This mode defines the token consumption and production rates on all channels and an execution time. See [15] for a complete introduction to SADF.

Using the performance analysis tool for SADF [9] which is part of the SDF3 tools from [16] it is possible to acquire performance metrics like *throughput*, *inter firing latency*, *response delay* and others. The *inter firing latency* is the minimum, maximum, or average time between consecutive firings of a particular actor, whereas the *response time* is the time until a particular actor initially fires.

IV. Template-based gesture recognition on multi-core architectures

Before we present our proposed SADF model of the hand gesture detection system, we introduce the application in more detail. Gesture recognition, like activity recognition is typically performed in several stages. Those stages form a so called

Activity Recognition Chain (ARC). In the following, we will explain the ARC stages of a typical template based gesture recognition system in the order as described in [17].

In the *Data Acquisition* stage, sensor data is sampled with a certain frequency. In the *Preprocessing* stage filters or corrections can be applied to the sensor signals. After preprocessing the continuous incoming data is segmented in order to perform online gesture detection. A typical approach for this is the usage of a sliding window. Thereby temporal snapshots of a certain size of the continuous data, so called *windows*, are considered for a gesture recognition. Since windows can overlap, the windows are referred to as *sliding windows*. This process is done at the *Segmentation* stage of an ARC. In the *Feature Extraction* stage, feature vectors are extracted from the data segments. Those are typically statistical properties or results from additional signal transformation.

In template-based gesture recognition, a signal or feature vector of a currently performed gesture is compared to a set of templates at the *Modeling and Inference* stage. The template which most resembled the signal or feature vector most likely represents the performed gesture. Those templates can be extracted features or the raw sensor signals of earlier performed gestures or of modeled gestures. In our implementation we use the *Dynamic Time Warping* (DTW) algorithm to perform the template matching.

In the last stage, the *Classification* is performed based on the results of former stages. A template-based gesture recognition system usually classifies the data within the current segment with the label of that template which showed the greatest similarity to that segment. Therefore the template with the greatest similarity to the sliding window data is evaluated and the sliding window is labeled with this gesture.

Since the *Modeling and Inference* stage is the most computation intensive task, an optimization regarding the computation time is done to perform gesture recognition on mobile devices efficiently. In [7] a configurable approach for using two different kinds of parallelism within the *Modeling and Inference* stage of template-based gesture recognition has been introduced to utilize multiple processor cores for this task. This approach can compromise between response time and scalability when applied on a homogeneous multi-core architecture.

As described in [7] an initiator-worker structure is deployed for the *Modeling and Inference* stage and the group of worker cores is subdivided into equal sized groups called *tiles*. In the following, the number of tiles in which the worker cores are subdivided is called a *configuration*. Figure 1 shows the *configuration* 3, where the system is using three tiles with two cores per tile. As a new sliding window from *Segmentation* stage arrives, the initiator core sends it to a free tile. A tile is responsible for the computation of the likeness to all templates of a sliding window. The set of gesture templates is distributed over all cores belonging to a tile, such that the data in the window can be compared to multiple templates simultaneously.

In a mobile context it is necessary to save as much energy as possible. Therefore it is important to avoid gratuitous

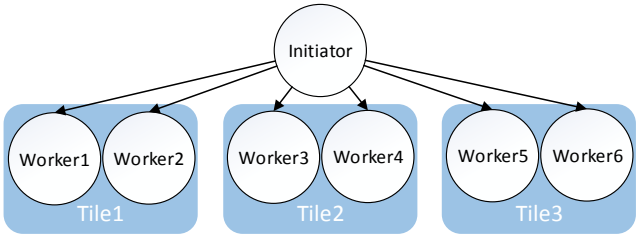

Fig. 1. Initiator-Worker structure with three tiles

computations, since energy consumption is directly related to hardware utilization. To avoid unnecessary gesture recognition for segments in which no activities are captured, we propose a measurement of the fluctuation of the signal at the feature extraction stage. If the magnitude of this signal change is below a certain threshold, the segment could be discarded and computation time, thus energy could be saved in the following stages. Since the gesture recognition searches for windows of high activity, the change of the signal is filtered with a basic *exponential moving average* (EMA) filter. This method has already been used to recognize the begin and end of activities in [18].

Additional to the recognition pipeline described in [7] we introduce the stage responsible for computing the *exponential moving average* (EMA) of the changes between all samples within the window. In order to speed up this task, we implemented the calculation without computing the square root. Equations (1) and (2) mathematically describe this calculation which is performed on a whole sliding window, where x_k, y_k, and z_k are the k-th sampled acceleration value from dimension x, y, and z, respectively. The smoothing factor α which is 0.2 in our implementation has been adopted from [18].

$$H_k = (x_k - x_{k-1})^2 + (y_k - y_{k-1})^2 + (z_k - z_{k-1})^2 \quad (1)$$

$$EMA_{H_k} = \alpha H_k + (1 - \alpha) EMA_{H_{k-1}} \quad (2)$$

Based on the value computed for a sliding window, the next sliding window is processed in the same way, or the gesture recognition is skipped for that window and just the exponential moving average is computed. This behavior will later be captured using scenarios in the SADF model. Since this computation only has to be performed once for each sliding window, we decided to only let the last core of a tile compute this value, because our experiments showed that the heuristic of [7] for distributing the set of templates over the cores of a tile usually leads to the last core having the least amount of data to process.

V. SADF Model of the Gesture Detection Algorithm

The extended dataflow model of the gesture detection system introduced in [7] which is based on EIDF has no capabilities to capture timing information. In the paper at hand, we introduce Scenario-Aware Dataflow (SADF) as model of computation, since it can capture both timing information of actors in form of execution times and different execution times, production, and consumption rates of actors based on its scenarios. We

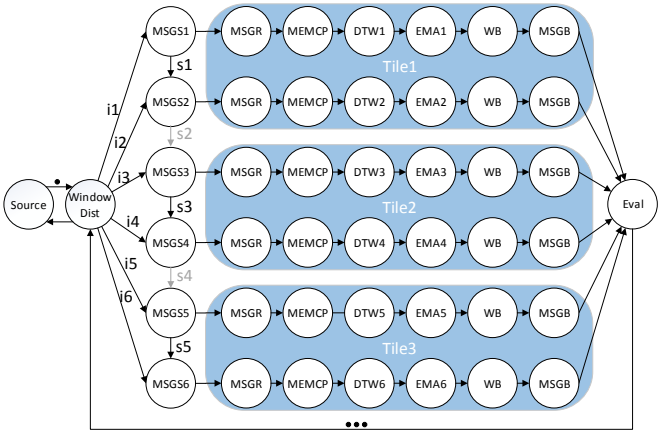

Fig. 2. SADF structure of the gesture recognition system in configuration 3

propose to use two different scenarios which model an enabled and a disabled *Modeling and Inference* stage, based on the EMA of the preceding sliding window.

As described in [7], the calculations on different cores are modeled by independent chains of actors which fire sequentially when executed on the same processor core, but actors of different chains can fire simultaneously like executed on different processor cores. The structure of our SADF model is shown in Fig 2, where we omitted the detector for the sake of simplicity. The `Source` actor represents the `Segmentation` stage which provides a new sliding window with a fixed frequency. The `WindowDist` actor sends those windows to different tiles in a cyclo-static manner based on the configuration, which means the number of tiles. For example: if six worker cores are available and we use a configuration of three tiles, we have two cores per tile. In our model, this would lead to `WindowDist` actor having three scenarios in which it can act. In scenario 1 it produces a token on the input edges `i1` and `i2` of the first two actor chains belonging to `Tile1`, which represents the process of sending the sliding window to those cores. In scenario 2 the `WindowDist` actor would produce tokens on channel `i3` and `i4` and in scenario 3 on channel `i5` and `i6`, respectively. In our implementation the *Initiator* DSP core sends messages containing a pointer to the sliding window data to the corresponding worker cores. Since the process of sending messages is done sequentially those processes are captured in the actors `MSGS1` to `MSGS6`, which are connected through the synchronization edges `s1`, `s2`, `s3`, `s4`, and `s5`. Those edges ensure the sequential execution of actor `MSGS1` and `MSGS2`, when the current sliding window is sent to the first tile. On the synchronization edges between actors of different tiles, production and consumption rates are 0. For example: in a configuration of three tiles, production and consumption rates to and from edges `s2` and `s4` are 0, which is illustrated by greyed out edges in Fig. 2.

The `MSGR` actors capture the time of receiving the messages from the *Initiator* core and `MEMCP` represent the process of copying the sliding window data from shared memory to the scratchpad memory of the particular DSP core. The `DTW` actors are used to model the Dynamic Time Warping computation between the sliding window and the template gestures. Note that the execution times of `DTW` actors differ on different cores within a tile, since in our case templates of different sizes have been distributed over those cores. The `EMA` actors capture the process of calculating the EMA of a sliding window. The cores on which EMA calculations are performed depend on the configuration. In order to keep the regular structure of the SADF graph, regardless which configuration is used, every actor chain contains an `EMA` actor but on those cores not computing the EMA, this actor has an execution time of 0. Actor `WB` is used to capture the time for writing back the similarity results between sliding window and templates to shared memory. Actor `MSGB` represents the process for sending a message with a pointer to the similarity results to the evaluating DSP core, which is modeled by the `Eval` actor. This core collects the results and performs the classification based on the similarities. After that it sends synchronization messages to the initiator core which indicates that the tile which processed the specific sliding window is now ready to calculate a new sliding window. This is modeled with a feedback channel from actor `Eval` to actor `WindowDist`.

Based on the result of the exponential moving average calculation of the preceding window, two scenarios, namely a result over a specific threshold (*highEMA*) or below this threshold (*lowEMA*) are defined. Each actor belonging to a tile in Fig. 2 can act in those scenarios.[1] Since only an EMA calculation is performed in a *lowEMA* scenario on the last core of the tile the execution time of actors of all other cores of that tile are 0 in scenario *lowEMA*. In Fig. 3 this is indicated by greyed out actors. Only the actors of the last core of a tile in scenario *lowEMA* are assigned with their corresponding execution times, except the `DTW` actor which also has an execution time of 0, since no gesture recognition is performed in this scenario. In scenario *highEMA* all actors of a tile are assigned with execution times bigger then 0, except for most of the *EMA* actors, for the same reason as already described in the *lowEMA* scenario. Note that the markov chain shown in Fig. 3 is only a part of the markov chain of our SADF model. Its transition annotations model the probability of occurrences of fluctuations in the sensor signals.

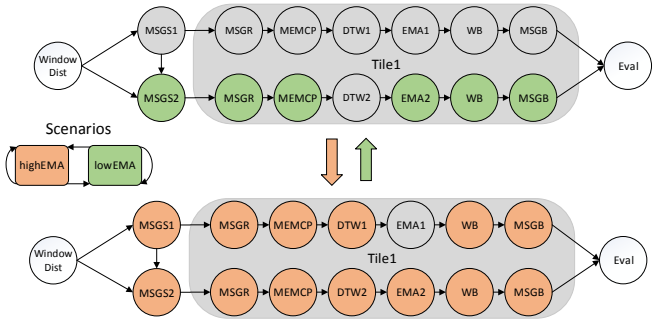

Fig. 3. Actors of the first tile of configuration 3 in scenario *lowEMA* (top) and *highEMA* (bottom)

[1]Note that a third scenario is assigned to all actors which is introduced to capture the 'inactivity' of a worker core. This Scenario will be explained in more detail in section V-A

In order to obtain a cyclo-static behavior sending sliding windows to tiles and receiving their results, actors `WindowDist` and `Eval` also act in different scenarios. Each scenario represents the distribution of one window to exactly one particular tile, while all other tiles do not receive that window. Those scenarios are triggered in a cyclo-static manner. In Fig. 4 this procedure is shown for a configuration of three tiles. In scenario *SentTo3* actor `WindowDist` only produces a token on those input edges, which correspond to the 3rd tile. Equivalently, actor `Eval` consumes a token from this tile in its scenario *GetFrom3* which is synchronized with scenario *SentTo3* of `WindowDist`.

A. *The scenario* inactive

In our model the `Eval` actor is actually modeled as a detector, since in this stage the result from the `EMA` calculation for a particular sliding window is evaluated and determines the scenario of the worker core, which processes the next sliding window. This actor has to produce at least one control token to all controlled actors, which include all actors belonging to a tile in Fig. 2. Since the sliding windows are dispatched in a cyclo-static manner to different tiles, except for a configuration with 1 tile, the actors belonging to tiles fire fewer times than the `Eval` actor fires. For example: In a configuration with three tiles the `Eval` actor fires three times while the actors of the first tile fire one time, since a token is only produced every third firing of actor `WindowDist` to the actors `MSGS1` and `MSGS2`. This would lead to a buffer overflows on those control channels, since more tokens are produced from `Eval` than being consumed from controlled actors.

In order to keep those control channels bounded, an *inactive* scenario is introduced to all actors belonging to a tile. In the *inactive* scenario, no tokens are consumed/produced with an execution time of 0 time units. This enables those actors to consume a control token, by neither delaying any other following firings of those actors nor producing any tokens to synchronization edges. This practically means, that every time all actors of a tile get a control token to select the *highEMA* or *lowEMA* scenario, additional control tokens are produced for all actors belonging to other tiles to trigger the *inactive* scenario. This procedure is shown in Fig. 4. Dashed arrows indicate control channels occupied by control tokens produced by the detector node. Although contrary to the other pictures of our model, in Fig. 4, the `Detector` is shown as a separate actor and only the cyclo-static part of its markov chain is illustrated for the sake of clarity. In our complete model (not illustrated in this paper), the markov chain of the detector `Eval` is actually a product of both markov chains, shown in Fig. 3 and Fig. 4.

Introducing an *inactive* scenario also has a positive effect to the calculation of the hardware utilization, which is explained in the following section.

B. *Processor Utilization*

We have extended the performance analysis tool for SADF in SDF3 [11] with the ability to compute the scenario occurrence probabilities for each actor. In case an actor can only operate in a single scenario (alike SDF actors), this occurrence probability trivially equals 1. For actors that can operate in multiple scenarios, the tool extension generally exploits the performance model checking approach as described in [10]. However, in case the scenario occurrences depend only on the behavior of a single Markov chain, the more efficient approach of deriving the scenario occurrence probabilities from the equilibrium distribution of that Markov chain is applied. We remark that an SADF graph with a single detector classifies for using this efficient approach for all actors.

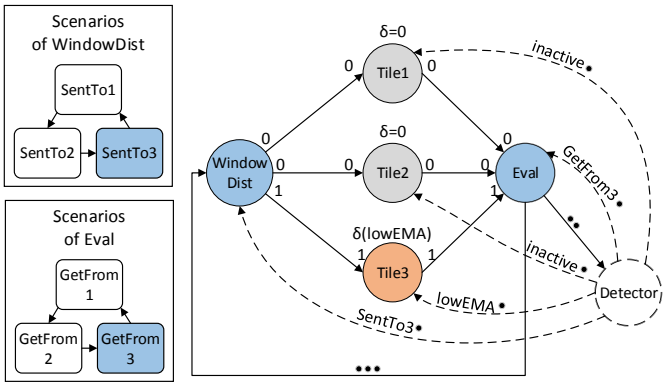

Fig. 4. Illustration of Tiles 1 and 2 being inactive, while Tile 3 is performing *lowEMA*.

This new feature enables us to calculate the ratio of computation time of a processor core within an observed time interval to the length of that interval, which corresponds to the processor utilization. Therefore consecutive actors without external dependencies are virtually mapped onto a processor core. We deduce this time interval from the difference of the arrival times of two consecutive sliding windows at a particular core. In a configuration with N tiles this period equals the *inter firing latency* (IFL) of the `Source` actor multiplied by N, since only the N-th sliding windows are dispatched to the same core. We can use the IFL of actor `Source`, because this actor is synchronized with `WindowDist` by a feedback channel from `WindowDist` to `Source`. Since all actors mapped to a processor core always fire consecutively synchronized by the feedback channel from `Eval` to `WindowDist`, the computation time of the processor core is the cumulative average execution time of those actors. The cumulative average execution time CAE is calculated with equation Eq. 3 with M being the number of actors a_{ic} mapped onto a specific core c and S_i being the number of scenarios of that specific actor.

$$CAE = \sum_{1<i\leq M} \sum_{1<j\leq S_i} p_i(j) \cdot \delta_i(j) \tag{3}$$

The execution time of actor a_{ic} in scenario s_j is denoted as $\delta_i(j)$ and $p_i(j)$ is the occurrence probability of scenario s_j for actor a_{ic} which we get from the extension we introduced.

In our case we want to estimate the utilization of the worker cores of our gesture recognition system. Thus all actors of each independent actor chain of our model are mapped onto a corresponding processor core. In a configuration with N tiles, those actors CAE is one N-th of the CAE just considering scenarios *lowEMA* and *highEMA*, since only the N-th firing of

that actors is not in a *inactive* scenario with an execution time of 0. Because the CAE is implicitly divided by N, the *inter firing latency* of the `Source` actor has not to be multiplied by N anymore. Therefore the average utilization AU of that processor core c is calculated by:

$$AU_c = \frac{CAE_c}{IFL_{Source}} \tag{4}$$

where IFL_{Source} is the *inter firing latency* of actor `Source`. By selecting a `Source` actor, virtually mapping consecutive actors to processor cores and using the introduced SDF3 extension, we can obtain the processors utilizations from our model in a generic way.

C. Execution Time Annotations

We have annotated our model with execution times based on measured values derived from our implementation. In order to avoid cache effects, we disabled caching for the worker cores. We implemented the gesture recognition system on a shared memory architecture, where data transfer between multiple cores is performed through shared memory synchronized by messages which contain meta information and pointer to the transferred data. Thus the times for sending sliding windows to a worker core (`MSGS`) and sending results to the core which evaluates the results (`MSGB`) are equal and independent of the scenario.

All actors of our model have been annotated with the worst-case execution times derived from measurements of the particular processes on the multi-core DSP. For the sake of simplicity, we merged the actors belonging to a tile in Fig. 2 to the two actors `DTW_Mn` and `EMA_Mn`. This can be done, because those actors always fire consecutively being synchronized by the feedback channel to `WindowDist`. The `DTW_Mn` actor includes the execution times of `DTW`, `MSGR`, and `MEMCP`, whereas the actor `EMA_Mn` includes the execution times of `EMA`, `WB`, and `MSGB` of their corresponding scenarios. The Table V show detailed annotated execution times of all actors in all scenarios. Note that *inactive* scenarios are excluded, since the execution time of all actors in this scenario is 0.

VI. Experiments

We implemented a hand gesture recognition system which has already been introduced in [19] and [7]. Gestures are performed with a glove equipped with 3 accelerometers. We defined 5 Gestures including a `grasping motion`, `swiping left` and `swiping right` with two fingers and drawing a circle `clock-wise` and `counter clock-wise` with the index finger. To each of those gestures we recorded 5 reference recordings (templates) leading to a total of 25 differently sized reference recordings from 57 to 123 samples. Those reference recordings have been recorded at a frequency of 25 Hz. The gesture recognition system was implemented for a TMDSEVM6678L board from Texas Instruments [20]. It is equipped with the homogeneous multi-core processor TMS320C6678 with 8 Digital Signal Processor cores with a frequency of 1 GHz. Our implementation uses the SYS/BIOS real-time kernel [21] provided by Texas Instruments.

In our experiments, we performed offline tests in order to get reproducible results. Therefore we recorded test sequences of gestures on which the gesture recognition task is performed offline. In order to simulate online behavior, the sliding windows are dispatched to each tile, triggered by a clock module, which throws an interrupt at a certain frequency. To verify real-time behavior, we checked in our experiments if sliding windows had to be skipped due to no free tiles.

Since we disabled caching for the worker cores of our experiments the execution time of our system is increased approximately by the factor two. Because of this we doubled the period between sliding windows in comparison to earlier publications to 40 ms by shifting it by two samples instead of one. The sliding window itself has a fixed size of 125 samples which corresponds to 2.5 seconds.

In order to get the processor utilization from our implementation, we used the *System Analyzer Tools* [22] from Texas Instruments to monitor the load of the threads which perform the tasks included in all actors belonging to a tile. The task load values which have been averaged over a period of 40 ms were acquired through JTAG at runtime.

A. Performance Estimation

We evaluated the differences between the analytically acquired timing properties of our SADF model and the timing behavior of the implementation. Therefore we measured the time to process sliding windows depending on the occurrence of an activity within a sliding window. The measured values are summarized in Table I. Note that the accumulated time of sending and receiving a message (10,722 ns) is subtracted from the shown times, since this additional time was included in the measurements, due to the measuring process itself. Table II shows the *Response Time* (RT) for computing a single sliding window acquired from our SADF model. In order to test our model for real-time requirements, we also evaluated the *inter firing latency* (IFL) of the `Source` actor. Since this actor is synchronized with the `WindowDist` actor, the *inter firing latency* should equal its execution time to meet real-time requirements. If the *inter firing latency* is higher than its execution time, the processing of a sliding window is not finished in time and the next sliding has to be skipped.

TABLE I
Measured response times for processing a sliding window depending on the configuration

highEMA				lowEMA			
Cfg.	∅RT $[ms]$	σ $[ms]$	#values	Cfg.	∅RT $[ms]$	σ $[ms]$	#values
6	224,142	0.075	2,959	6	0.072	0.000054	8,734
3	117,490	0.057	2,695	3	0.072	0.000041	8,979
2	76,995	0.045	2,370	2	0.072	0.000040	8,231
1	40,564	0.016	1,287	1	0.072	0.000039	7,710

Additionally, Table II shows the difference between calculated and the corrected measured response times.

Comparing the measured *response times* and the analytically acquired *response times*, we can see that there is a small deviation for scenario *highEMA* of approximately 0.85 % for the configuration with one tile and 0.79 % for all other configurations. In scenario *lowEMA* there is a deviation of

TABLE II
RESPONSE TIMES AND INTER FIRING LATENCIES ACQUIRED FROM THE SADF MODEL

highEMA				lowEMA			
Cfg.	RT $[ms]$	IFL $[ms]$	ΔImpl.	Cfg.	RT $[ms]$	IFL$[ms]$	ΔImpl.
6	222.387	40	0.79%	6	0.061955	40	13.59%
3	116.568	40	0.79%	3	0.061955	40	13.35%
2	76.394	40	0.79%	2	0.061955	40	13.36%
1	40.220	40.2203	0.85%	1	0.061955	40	13.36%

approximately 14 %, which is much higher than in scenario *highEMA*. It leads to the assumption that there is an additional absolute error apart from a relative error between model and implementation. However, such an error has much less impact on the *highEMA* scenario since the *response delay* in that scenario is much higher than in scenario *lowEMA*. Since the real-time requirement in scenario *highEMA* is the important one, this error is acceptable to deduce the real-time ability of different configurations. From the analytically acquired properties shown in Table II we predict that the configuration with one tile does not meet real-time requirements in scenario *highEMA*, since the IFL of actor `Source` exceeds 40 ms. We could verify that in our experiments, because every other sliding window is discarded in scenario *highEMA* due to the exceeding processing time of that sliding windows. All other configurations meet real-time requirements which are in line with the analytically acquired results from the SADF model.

We also calculated the utilization of all worker cores based on the results from the SADF model. In Table III the analytically acquired processor utilizations are summarized for all configurations.

TABLE III
ANALYTICALLY ACQUIRED PROCESSOR UTILIZATION FROM OUR SADF MODEL

highEMA					lowEMA			
Config	1	2	3	6	1	2	3	6
Core1	98.8%	95.5%	97.1%	92.7%	0.00%	0.00%	0.00%	0.02%
Core2	91.2%	95.0%	88.2%	92.7%	0.00%	0.00%	0.04%	0.02%
Core3	99.9%	87.6%	97.1%	92.7%	0.00%	0.06%	0.00%	0.02%
Core4	89.0%	95.5%	88.2%	92.7%	0.00%	0.00%	0.04%	0.02%
Core5	88.3%	95.0%	97.1%	92.7%	0.00%	0.00%	0.00%	0.02%
Core6	86.0%	87.6%	88.2%	92.7%	0.13%	0.06%	0.04%	0.02%

Very first results of the task load from the implementation are shown in Table IV. Except for configuration 1 all task load values of the implementation in scenario *highEMA* in Table IV show a relative error of less than 2 % to the analytically acquired utilizations from the SADF model shown in Table III. This indicates a high accuracy of the introduced approach estimating the utilization for those configurations. However, for configuration 1 in this scenario the task load of our implementation is approximately half of the acquired utilizations from the model. This can be explained by the fact that this configuration does not meet the real-time requirements, which leads to a discard of every other sliding window in the implementation. Contrary to this in our SADF model no sliding windows are discarded when exceeding 40 ms for processing a sliding window. Thus the task load of the implementation in configuration 1 is approximately half of the expected values acquired from the SADF model.

TABLE IV
TASK LOAD OF THE WORKER CORES IN BOTH SCENARIOS *highEMA* AND *lowEMA* ACQUIRED FROM OUR IMPLEMENTATION

highEMA					lowEMA			
Config	1	2	3	6	1	2	3	6
Core1	49.9%	96.2%	97.8%	93.3%	0.00%	0.00%	0.00%	0.13%
Core2	46.2%	95.6%	88.8%	93.3%	0.00%	0.00%	0.13%	0.00%
Core3	50.6%	88.2%	97.8%	93.3%	0.00%	0.13%	0.00%	0.00%
Core4	45.0%	96.2%	88.8%	93.3%	0.00%	0.00%	0.00%	0.00%
Core5	44.7%	95.6%	97.8%	93.3%	0.00%	0.00%	0.00%	0.00%
Core6	43.5%	88.2%	88.8%	93.3%	0.13%	0.00%	0.00%	0.00%

In scenario *lowEMA* there are also differences in the task load acquired from the implementation and the values from the model. The reason for this is that in our implementation the initiator core is not generally distributing sliding windows in a cyclo-static manner. Sliding windows are sent to the next free tile. In a *highEMA* scenario this in fact leads to a cyclo-static behavior, but with introducing *lowEMA* scenarios, the first tile completes the calculation before the next window arrives. Thus the initiator core sends the next sliding window also to the first tile, because it is free. So if only *lowEMA* scenarios occur only the first tile will process all incoming sliding windows, which is contrary to our SADF model. This causes the effect, that the utilizations of our model in scenario *lowEMA* are distributed over the last cores of all tiles, while in our implementation the utilization is accumulated to the last core of only the first tile. Summing up the utilization values from all EMA calculating cores in scenario *lowEMA* from Table III shows that the sum has an absolute deviation of 0.01 % from the task load of the EMA calculating core of the first tile in our implementation. This corresponds to a relative error of the predicted utilization of less than 8 %.

B. Recognition Performance

Before introducing the EMA stage to our algorithms, we evaluated the recognition performance of the recognition system. Note that recognition performance describes the correctness of the system distinguishing between gestures and non-gestures and how accurate it classifies them. We calculated the F1-score, which is the harmonic mean of the ratio of correct classifications to all classifications and the ratio of correct classifications to all performed gestures [17]. The F1-score of the recognition performance was 66.4 %. This is a rather mediocre result, but from a plain DTW recognition on raw sensor data, without any optimizations on the recognition accuracy. By introducing the `EMA`, the recognition system can save a lot of computation time on worker cores, but since the `EMA` detects the beginning and end of a gesture, the recognition performance is directly affected. By introducing the `EMA` the F1-score of the recognition performance fell to 62.2 % which is not a big decrease, but a similar mediocre result. However, since we focused on parallelization aspects and computation time we did no additional optimization regarding the recognition performance.

VII. Conclusions

In this paper we showed a Scenario-Aware Dataflow model for template-based gesture recognition on multi-core platforms with the ability to predict functional and extra functional properties. This model allows substantiating early design decisions as different parallelization approaches and configuration parameters for design space exploration. In our experiments we fitted and calibrated this system for the Texas Instruments multi-core DSP chip TMS320C6678. For the *response times* of the system our results show an error of less than 1 % for high computation load scenarios and less than 14 % error for low computation load scenarios. Since the high computational load scenario is the critical component for the further analysis of real-time capabilities and response time, the higher error of the low load scenario is acceptable. Furthermore the predicted utilization of the system shows an error of less than 2 % for high computation load scenarios and an error of less than 8 % for low computation scenarios to the actual task load acquired from the implementation. This metric helps to enhance the set of reference gestures for the desired application in an early development stage. Moreover, our model can be used to predict the real-time capabilities of different configurations. This is crucial for online gesture recognition, especially in the context of Cyber-Physical Systems.

TABLE V
Annotated execution times of all actors in all scenarios

highEMA				
Config	1	2	3	6
WindowDist	0	0	0	0
MSGS[1..6]	4432	4432	4432	4432
DTW_M1	39731366	76376876	116550836	222343926
DTW_M2	36666236	75960516	105813816	222343926
DTW_M3	40194686	70047986	116550836	222343926
DTW_M4	35786556	76376876	105813816	222343926
DTW_M5	35512076	75960516	116550836	222343926
DTW_M6	34556636	70047986	105813816	222343926
EMA_M1	5097	5097	5097	31056
EMA_M2	5097	5097	31056	31056
EMA_M3	5097	31056	5097	31056
EMA_M4	5097	5097	31056	31056
EMA_M5	5097	5097	5097	31056
EMA_M6	31056	31056	31056	31056
Eval	7254	7254	7254	7254
lowEMA				
Config	1	2	3	6
WindowDist	0	0	0	0
MSGS1	0	0	0	4432
MSGS2	0	0	4432	4432
MSGS3	0	4432	0	4432
MSGS4	0	0	4432	4432
MSGS5	0	0	0	4432
MSGS6	4432	4432	4432	4432
DTW_M1	0	0	0	20717
DTW_M2	0	0	20717	20717
DTW_M3	0	20717	0	20717
DTW_M4	0	0	20717	20717
DTW_M5	0	0	0	20717
DTW_M6	20717	20717	20717	20717
EMA_M1	0	0	0	30410
EMA_M2	0	0	30410	30410
EMA_M3	0	30410	0	30410
EMA_M4	0	0	30410	30410
EMA_M5	0	0	0	30410
EMA_M6	30410	30410	30410	30410
Eval	6396	6396	6396	6396

References

[1] K. Assaleh, T. Shanableh, and M. Zourob, "Low complexity classification system for glove-based arabic sign language recognition," in *Neural Information Processing*. Springer, 2012, pp. 262–268.

[2] A. H. Ali, A. Atia, and M. Sami, "A comparative study of user dependent and independent accelerometer-based gesture recognition algorithms," in *Distributed, Ambient, and Pervasive Interactions*. Springer, 2014.

[3] B. Choe, J.-K. Min, and S.-B. Cho, "Online gesture recognition for user interface on accelerometer built-in mobile phones," in *Neural Information Processing. Models and Applications*. Springer, 2010.

[4] D. J. Burr, B. Ackland, and N. Weste, "A high speed array computer for Dynamic Time Warping," *ICASSP'81*, pp. 471–474, 1981.

[5] Y. Bae and M. Fairhurst, "Parallelism in dynamic time warping for automatic signature verification," *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, 1995.

[6] M. Yoder and L. Siegel, "Dynamic time warping algorithms for SIMD machines and VLSI processor arrays," *IEEE ICASSP '82*, vol. 7, 1982.

[7] F. Grützmacher, J.-P. Wolff, and C. Haubelt, "Sensor-based online hand gesture recognition on multi-core dsps," in *Proceedings of the Symposium on Signal Processing on Graphics Processing Units and Multicores, Orlando, Florida, USA, December 2015*.

[8] W. Plishker, N. Sane, M. Kiemb, K. Anand, and S. S. Bhattacharyya, "Functional dif for rapid prototyping," in *19th IEEE/IFIP International Symposium on Rapid System Prototyping, 2008*. IEEE, 2008, pp. 17–23.

[9] B. D. Theelen, M. C. Geilen, T. Basten, J. P. Voeten, S. V. Gheorghita, and S. Stuijk, "A scenario-aware data flow model for combined long-run average and worst-case performance analysis," in *Proc. of MEMOCODE'06*. IEEE Computer Society, pp. 185–194.

[10] B. Theelen, M. Geilen, and J. Voeten, "Performance model checking scenario-aware dataflow," in *Formal Modeling and Analysis of Timed Systems*. Springer, 2011, pp. 43–59.

[11] B. D. Theelen, "A performance analysis tool for scenario-aware streaming applications," in *In Proc. of QEST'07*. IEEE, 2007, pp. 269–270.

[12] M. Geilen, S. Stuijk, and T. Basten, "Predictable dynamic embedded data processing," in *In Proc. of SAMOS'12*. IEEE, 2012, pp. 320–327.

[13] Y. Yang, M. Geilen, T. Basten, S. Stuijk, and H. Corporaal, "Playing games with scenario-and resource-aware sdf graphs through policy iteration," in *In Proc. of DATE'12*. IEEE, 2012, pp. 194–199.

[14] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sept 1987.

[15] S. S. Bhattacharyya, E. F. Deprettere, and B. D. Theelen, "Dynamic dataflow graphs," in *Handbook of Signal Processing Systems*. Springer, 2013, pp. 905–944.

[16] S. Stuijk, M. Geilen, and T. Basten, "Sdf ^ 3: Sdf for free," vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 276–278.

[17] A. Bulling, U. Blanke, and B. Schiele, "A tutorial on human activity recognition using body-worn inertial sensors," *ACM Computing Surveys (CSUR)*, vol. 46, no. 3, p. 33, 2014.

[18] Z. Prekopcsák, "Accelerometer based real-time gesture recognition," 2008.

[19] F. Grützmacher, J.-P. Wolff, and C. Haubelt, "Exploiting thread-level parallelism in template-based gesture recognition with dynamic time warping," in *Proceedings of the 2nd iWoAR*. ACM, 2015, p. 6.

[20] "TMDSEVM6678L EVM Technical Reference Manual," 2012, http://www.farnell.com/datasheets/1691442.pdf, literature number: SPRUH58.

[21] "SYS/BIOS (TI-RTOS Kernel) v6.45 User's Guide," 2015, http://www.ti.com/lit/ug/spruex3p/spruex3p.pdf, literature number: SPRUEX3P.

[22] "System Analyzer User's Guide," 2014, http://www.ti.com/lit/ug/spruh43f/spruh43f.pdf, literature number: SPRUH43F.