Triple-C: resource-usage prediction for semi-automatic parallelization of groups of dynamic image-processing tasks
Albers, A.H.R.; de With, P.H.N.; Suijs, E.

Published in:

DOI:
10.1109/IPDPS.2009.5160942

Published: 01/01/2009

Citation for published version (APA):
Abstract

With the emergence of dynamic video processing, such as in image analysis, runtime estimation of resource usage would be highly attractive for automatic parallelization and QoS control with shared resources. A possible solution is to characterize the application execution using model descriptions of the resource usage. In this paper, we introduce Triple-C, a prediction model for Computation, Cache-memory and Communication-bandwidth usage with scenario-based Markov chains. As a typical application, we explore a medical imaging function to enhance objects of interest in X-ray angiography sequences. Experimental results show that our method can be successfully applied to describe the resource usage for dynamic image-processing tasks, even if the flow graph dynamically switches between groups of tasks. An average prediction accuracy of 97% is reached with sporadic excursions of the prediction error up to 20-30%. As a case study, we exploit the prediction results for semi-automatic parallelization. Results show that with Triple-C prediction, dynamic processing tasks can be executed in real-time with a constant low latency.

1. Introduction

Advanced video processing is becoming increasingly dynamic, which poses new requirements on the system design. With dynamic video-processing applications, such as in image analysis, the computational complexity has become data dependent and memory usage is more irregular. Detailed know-how of specific application aspects, such as data-driven complexity and the corresponding memory requirements is relevant for optimal mapping of tasks on a computing platform and optimizing the performance during runtime.

In order to guide the mapping and obtain efficiency in implementation, performance prediction may be applied in the form of modeling. This paper concentrates on achieving sufficient accuracy in the modeling for applications featuring dynamic execution of tasks.

The secondary objective is to use the modeling for runtime estimation of the resource usage with the aim to execute more functions on the same platform. The model descriptions are used as a prediction for resource planning, parallelization and possibly the corresponding Quality-of-Service (QoS) control [1][2].

The above problem statement has been addressed in several fields of applications, such as high-performance computing and multimedia. Within performance computing, several techniques are reported in the literature for performance prediction of parallel applications [3][4][5]. Most of them are based on block-parallel or dataflow-like approaches. The drawback of these approaches is that the analysis is restricted to applications which have static processing requirements at known rates and data sizes, while the required computations have a deterministic nature.

In the multimedia domain, application analysis has revealed that algorithms have variable rates and sizes, which involves application tuning based on this variable nature. In such a case, the application is described by its key features such as a simple average or discrete distributions acquired from profile analysis [6][7]. In our case, the application field is professional medical imaging and processing. Since this involves a rather broad scope, we study a mixture of the previous features and new aspects. A difference is that our application involves image analysis, which has a more dynamic nature and will be discussed below.
The paper is organized as follows. In Section 2, we start with a survey of the system and architectural requirements in our application field. Section 3 presents the medical-imaging application under study. Performance modeling of the video-processing functions, memory communication and granularity optimization is discussed in Section 4 and 5. For runtime estimation of the resource usage, in Section 6, we introduce task-level scalability and runtime quality control of the application. Section 7 presents experimental results for mapping the image-analysis application on a multiprocessor platform with a constant low latency. Section 8 concludes the paper.

2. System and Architectural Requirements

The work presented in this paper aims at realizing a design flow involving the mapping and runtime management of advanced video applications onto a multiprocessor platform. We focus on the upper design layers, dealing with the applications and their control for an efficient execution. Let us start with a survey of the system and architectural requirements which are important for our application field in professional medical imaging.

- **Low latency.** We explore a medical imaging function to enhance objects of interest under X-ray fluoroscopy imaging, during a live interventional angiography procedure [8][9][10]. Because physicians must see their actions directly on the screen (eye-hand coordination), a constant low latency is a key requirement for the real-time imaging application.
- **Off-the-shelf multiprocessor.** In many medical imaging procedures, a multitude of imaging functions is carried out in parallel. Because dedicated hardware lacks flexibility, using off-the-shelf multiprocessors as a platform for software image-processing seems to be a valid choice. The selected platform contains multiple processing cores in order to provide the required processing power, while maintaining some degree of runtime programmability and flexibility. To increase the cost-efficiency, specialistic co-processors (such as graphics cards) can be added in the future.
- **Variable image analysis.** A trend in medical imaging is the introduction of image analysis and feature extraction techniques to increase the performance of other tasks in the real-time video pipeline (like image enhancement). From an architectural perspective, the computational complexity has become data dependent and memory usage is more irregular. As an answer to the variable processing rates, performance prediction may be applied in the form of modeling to guide the mapping and to obtain an efficient implementation. Modeling of application resource usage is complicated because depending on the image content and intermediate analysis result, the analysis algorithm may switch to a different group of processing tasks.

In this paper, we introduce Triple-C, a prediction model for Computation, Cache-memory and Communication-bandwidth usage for groups of dynamic image-processing tasks with scenario-based Markov chains. It is important to note that our approach is essentially different from considering only the worst-case state of the system. Unlike the worst-case approach, our approach is dynamic, i.e. it makes use of runtime characteristics of the input data and the environment of the application, with the aim to execute more functions on the same platform.

3. Medical image-analysis application

Let us now discuss the key application used to evaluate our architectural study. Coronary angioplasty is a catheter-based procedure performed by an interventional cardiologist in order to open up a blocked coronary artery and restore blood flow to the heart muscle. Angioplasty is used as an alternative treatment to coronary artery bypass surgery in more than half of the cases. Following balloon angioplasty, a wire mesh tube (stent) can be placed to keep the artery open. The correct deployment of a stent in the coronary arteries is important for ensuring the efficiency of drug-eluting
Motion-compensated feature enhancement consists of several steps, as depicted in Figure 2. The presented flow graph is based on a cascading of four stages which are individually described in [11][12][13][14]. After stent placement, the candidate balloon markers are detected in the image using an automatic marker-extraction algorithm. Ridge detection (RDG) and filtering is applied to the input images such that all other structures, except candidate balloon markers, are removed. If no other dominant structures are detected, the function is skipped. Subsequently, marker extraction (MKX EXT) selects punctual dark zones contrasting on a brighter background as candidate markers.

Based on a-priori known distances between the balloon markers, couples selection (CPLS SEL) selects the best marker couple from the set of candidate couples. Subsequently, temporal registration (REG) to align respective markers in selected image frames, is based on a motion criterion, where a temporal difference is performed between two succeeding images of the sequence. A Region Of Interest is estimated in the original image (ROI EST), where the markers have previously been detected. The guide wire can be detected by a ridge filter in guide-wire extraction (GW EXT). If the markers of a possible couple are situated on a track corresponding to a ridge joining them (the guide wire), this is the indication that the results obtained by automatic marker extraction are found stable. Enhancement (ENH) of the stent is performed by temporal integration of the registered image frames according to the balloon markers. The output is presented by zooming (ZOOM) in the ROI containing the stent. As evident from the images in Figure 1, the enhanced images enable an improved control of the good expansion of the stents and of their positioning relative to the vessel wall without the use of other devices, such as intravascular ultrasound.

The described application is dynamic in three major aspects: (1) At the start, an ROI of variable data-dependent size is chosen for further analysis, and (2) at every stage, switch functions select a specific flow graph, depending on the previous stage(s). Moreover (3), some of the internal flow graphs require intrinsically a variable processing time. Performance prediction is challenging as the input data may cause variability within task execution times. Moreover, the dynamic decision making is based on outcomes of the image analysis process which heavily depend on the input video data. Tasks in the image analysis cannot be easily switched off, since that would lead to an incomplete or unacceptable result. In the next section, we introduce the prediction model for the computation time including the estimation technique coping with the dynamic behavior.

4. Prediction Model of Computation Time

As an answer to variable processing rates of the application, performance prediction may be applied in the form of modeling to guide the mapping and to obtain an efficient implementation. Modeling of application resource usage is complicated because depending on the input data and intermediate analysis result, the analysis algorithm may switch to a different group of processing tasks.

We have considered several options for modeling of the computation time. As a first solution, we investigated literature on video traffic modeling [15]. Most of the papers deal with Markov-chain approaches since the estimation of the model parameters is straightforward and there is a large number of analysis techniques available.
A Markov chain is a stochastic process with the Markov property. The Markov property means that, given the present state, future states are independent of the past states. In other words, the description of the present state fully captures all the information that could influence the future evolution of the process. Future states will be reached through a probabilistic process instead of a deterministic one.

To deal with applications for which the computation time depends on long-term statistics of the video frames, higher-order probabilistic processes can be used, but the state space will grow exponentially. Also, a problem is to obtain statistically significant estimates for the transition probabilities, because with an increasing order, the number of samples for each estimate is very small, even for long data sets. An alternative view on the modeling of the system behavior is to consider the timing statistics of the video frames in two categories, as a result from mapping the algorithms on a platform. Hence, we then investigate short-term and structural fluctuations in processing time on the platform. Short-term fluctuations can be caused by cache misses or the overhead imposed by task switching and control. Structural fluctuations are caused by the dependency of the processing time of the tasks on the video content itself over a longer time period. This points to the direction of splitting the computational statistics in categories.

As a consequence of the previous discussions, we have adopted a concept where the long-term statistics are decoupled from the short-term stochastic behavior by employing different models for those statistics.

- **Short-term data correlations.** We try to describe the prediction model for the short-term data-dependent tasks as a probabilistic process such as a finite state Markov chain. A first-order Markov chain is by definition memoryless, where in the model it is implicitly assumed that the processing times of successive frames are independent. Based on computation of the autocorrelation function, we have concluded that couples selection (CPLS SEL) and guide-wire extraction (GW EXT) tasks can both be modeled with Markov chains. A disadvantage of Markov-chain modeling is the required exponentially decaying autocorrelation function of the input data. Markov-chain prediction falls short if processing times between video frames are correlated over a longer time period. Next, we will describe the modeling of long-term structural data dependencies.

- **Long-term data correlations.** We consider the prediction model to consist of long-term low-frequency fluctuations, around which short-term high-frequency fluctuations can take place. Discrimination between the low and high-frequency part can be made by various types of filters, such as Finite Impulse Response (FIR) or Infinite Impulse Response (IIR) filters. We apply the Exponentially Weighted Moving Average (EWMA) filter. As this IIR filter weights recent inputs more heavily than long-term previous ones, it adapts more quickly to the input signal compared to FIR filters. The EWMA filter is defined by:

\[ y(t_k) = (1 - \alpha) \times y(t_{k-1}) + \alpha \times x(t_k) \]  

(1)

Given the separation of correlation behavior, the short-term fluctuations are modeled with Markov chains. We have validated the applicability of the Markov-chain modeling by analyzing the autocorrelation function. In Fig. 3, the computation-time statistics for the ridge-detection (RDG) task are shown. To model the computation time for the current video frame, the output of the EWMA filter is used for long-term behavior prediction. On top of that, a Markov chain predicts the short-term fluctuations in computation time. The state-space description can be generated by analyzing the computation time over a long time period. The number of states \( M \) is \( C_{\text{max}}/\sigma_C \), where \( C_{\text{max}} \) denotes the largest measured value and \( \sigma_C \) the standard deviation. We have experimentally evolved to a model with approximately \( 2M \) states to obtain sufficient accuracy. The quantization intervals are adaptively chosen such that each interval contains on the average the same amount of samples. The entries of the transition probability matrix \( \{P_{ij}\} \) are estimated by

\[ P_{ij} = n_{ij}/(\sum_{k=1}^{M} n_{ik}) , \]  

(2)

where \( n_{ij} \) denotes the number of transitions from interval \( i \) to interval \( j \). Processing-time statistics for different Region-Of-Interest (ROI) sizes show that the RDG task has a linear dependency on the size of the ROI (See Fig. 6). To analyze load fluctuations, caused by dependencies on the video content itself, we have
subtracted a linear growth function from the obtained statistics. This function is specified by

\[ y_{\text{lin}} = 0.067 \times t_k + 20.6. \]  

(3)

For the remaining data-dependent fluctuations after subtraction, we analyzed the autocorrelation function. As the function has an exponential decay, it can again be described with a Markov chain. As the fluctuations are in the same order as the high-frequency behavior described in the previous section, we have included these statistics to the Markov state-generation process, to generate a single Markov chain for the ridge-detection task.

5. Cache-memory and Communication-bandwidth Analysis

In this section, we focus on the detailed analysis of the cache-memory and communication-bandwidth allocation for inter-task and intra-task communication. Optimal resource utilization following only the prediction model for computation time, usually conflicts with the usage of communication resources. A reliable reasoning about tasks having variable processing rates requires analytical methods, covering on one hand the actual required computation time, and on the other hand, the communication and storage requirements. We adopt this approach in Triple-C, in which, apart from prediction models for Computation time (Section 4), we also introduce analysis for Cache-memory and Communication-bandwidth usage. Let us start by describing the memory requirements for each task in the graph of Fig. 2, followed by the calculation of the communication bandwidth under various conditions.

5.1. Task Memory Requirements

The required amount of memory for each task can be derived by extracting the input/output requirements and intermediate storage requirement from a reference software implementation. In Table 1, the results are shown for the RDG, MKX, ENH and ZOOM task. For concentrating on significant bandwidth requirements, only operations on arrays are taken into account. The tasks that operate on a subset or feature data are negligible in terms of memory consumption. Note that some of the functions have different memory requirements, depending on the state of a switch in the flow graph. For example, if the RDG task is switched off, the succeeding MKX function has a much smaller input buffer requirement. The amount of communication bandwidth required for correct operation of the image-analysis application is derived by a mapping of the memory requirements onto the platform architecture.

<table>
<thead>
<tr>
<th>Task</th>
<th>RDG select Input (KB)</th>
<th>Intermediate (KB)</th>
<th>Output (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDG FULL</td>
<td>2,048</td>
<td>7,168</td>
<td>5,120</td>
</tr>
<tr>
<td>RDG ROI</td>
<td>2,048</td>
<td>5,120</td>
<td>5,120</td>
</tr>
<tr>
<td>MKX FULL</td>
<td>-</td>
<td>512</td>
<td>2,560</td>
</tr>
<tr>
<td>MKX ROI</td>
<td>-</td>
<td>512</td>
<td>2,560</td>
</tr>
<tr>
<td>MKX FULL</td>
<td>x</td>
<td>4,608</td>
<td>2,560</td>
</tr>
<tr>
<td>MKX ROI</td>
<td>x</td>
<td>4,608</td>
<td>2,560</td>
</tr>
<tr>
<td>ENH</td>
<td>2,048</td>
<td>8,192</td>
<td>1,024</td>
</tr>
<tr>
<td>ZOOM</td>
<td>1,024</td>
<td>4,096</td>
<td>4,096</td>
</tr>
</tbody>
</table>

Table 1. Memory requirements for each task of Fig. 2.

5.2. Communication-bandwidth Requirements

To calculate the required amount of communication bandwidth of the application under study, there are a number of constraints that influence the actually measured communication bandwidth load on the platform architecture.

- **Hardware platform.** For the experiments, we use a general-purpose multiprocessor platform containing two quad-core processors. In Figure 4, the system is shown. In total, the system consists of 8 processors of 2.33 GCycles/s, 8 level-1 caches of 32 KB and 4 level-2 caches of 4 MB. The system is equipped with 4 GB of external memory. For more details about the instantiated architecture, we refer to literature [16].

- **Application scenarios.** The application under study processes images of 1024×1024 pixels (2 Bytes/pixel) at 30 Hz. Due to the switch statements in the flow graph of Figure 2, there are multiple application scenarios possible. For the worst-case scenario in terms of bandwidth requirements, the tasks operate on a full-frame granularity, the ridge-detection task is activated and the registration phase completes successfully. On the opposite, for the best-case scenario in terms of bandwidth requirements, the tasks operate on a small
region-of-interest granularity, which will save a significant amount of required bandwidth. Furthermore, if the ridge-detection task is not required and the registration is not successful, the enhancement and zoom tasks will be skipped, which will save another significant amount of required bandwidth. Note that in this ‘best-case’ scenario in terms of bandwidth, the algorithm will not output a satisfying result. In total, there are eight different scenarios possible given the three switch statements in the flow graph.

- **Intra-task memory.** For several tasks, the required amount of intra-task memory will exceed the available local cache memory of the hardware platform. For the application under study, the RDG FULL, ENH and ZOOM tasks have an intra-task memory requirement that is higher than the level-2 cache capacity of the hardware platform. Therefore, additional communication bandwidth will be initiated to swap data in and out the external memory. The amount of bandwidth of each task can be modeled by analysis of the access pattern of the internal buffers. As the above tasks operate on a whole image granularity, scanning of the pixels linearly in the (x, y) direction will enable that all data items will be accessed. The modeling of the cache-memory occupation and corresponding eviction of internal buffers can be described with a space-time buffer occupation model. In Figure 5, for each subtask of the RDG function, the eviction of cache lines in case of overload is shown, as it requires extra bandwidth between cache memory and external memory storage.

- **Inter-task memory.** In Figure 2, the required inter-task bandwidth is shown on the arrows between tasks (1024 x 1024 pixels, 30 Hz, 2 Bytes/pixel). The mapping of the application on the platform has a direct relationship with the actually measured communication-bandwidth between tasks.

Figure 4. (a) Generic architecture model, (b) instantiated architecture with parameters.

Figure 5. Required intra-task bandwidth for RDG FULL task due to the limited cache-memory storage.

### 6. Semi-Automatic Parallelization

In this section, we want to exploit the modeling results from the previous sections for runtime estimation of the resource usage, as our aim is to execute more functions on the same platform. The model descriptions are used as a prediction for resource planning, parallelization and quality control.

For the application under study, data-dependent switch statements (Fig. 2) can cause the total processing time to change rather abruptly. As a result, the latency at the output may vary over time. However, during a live interventional X-ray procedure, large latency differences between succeeding frames are not allowed for clinical reasons (eye-hand coordination of the physician).

A straightforward solution is to employ a task partitioning on the platform, based on worst-case resource usage. With a delay function at the end of the pipeline, the output latency can be kept constant. As this solution will obviously work as expected, the main drawback is that for most of the time, the reserved resource budget is set too conservative. Furthermore, the output latency is higher than actually required. Moreover, it is impossible to exploit the difference between average-case and worst-case requirements without affecting the reliability (guaranteed performance) of the application.

Another approach is to use the prediction results from Triple-C for semi-automatic parallelization of the application. It is important to note that this approach is essentially different from considering the worst-case state of the system. Unlike the worst-case approach, our approach is dynamic, i.e., it makes use of runtime characteristics of the input data and the environment of the application. Using Triple-C, we are able to
accurately predict how much resources are required. This information can be used at runtime for on-the-fly repartitioning of groups of tasks. Furthermore, we have the possibility to execute more functions on the same platform. The approach consist of several steps.

- **Initialization:** By processing the first frame of the sequence, we initialize the partitioning of the flow-graph based on the image characteristics. The output latency is set to an initial value (close to average case), which will be our latency budget during runtime.

- **Runtime adaptation:** Based on the outcome from the resource predictions for subsequent frames, the resource manager can decide to repartition the flow-graph to handle an increase or decrease of resource consumption, to keep the output latency stable at the initialized (average-case) value.

- **Profiling:** The application can be profiled to gather statistical information of the differences between the actually consumed resources and the predicted values. The information can be used for on-line model training, or to give insight information about the prediction quality of the model descriptions.

For the application under study, the data of the RDG FULL and RDG ROI tasks can be easily partitioned, as the tasks have a streaming nature. For the CPLS SEL and GW EXT tasks, functional partitioning is more appropriate, as the tasks operate on extracted features, rather than image data. For a comparison between data-parallel partitioning and function-parallel partitioning, we refer to [17].

7. Experimental Results

In this section, we evaluate the analysis and prediction models on a set of test sequences. Furthermore, the statistics are used to guide the semi-automatic parallelization of the application, following the concept as explained in the previous section. Let us start by evaluating the accuracy of Triple-C.

The cache-memory and communication-bandwidth analysis from Section 5 are used for the prediction of memory and bandwidth usage for the set of test sequences. At a scenario level, the memory resource usage is more or less constant. The size of the ROI only slightly impacts the memory usage, therefore the differences between a large ROI and a small ROI are negligible in terms of bandwidth. For the test sequences, an average prediction accuracy between the analysis and measured cache-memory and communication-bandwidth usage of 90% is obtained.

Computation time statistics are obtained by profiling the executed application on a chip-multiprocessor platform. For the prediction of computation time for data-dependent tasks, we have applied probabilistic models based on known methods for modeling video traffic performance in networks. For tasks with long-term dependencies on input data frames, we have applied filtering to make the signal suitable for probabilistic modeling. Data-dependent switch statements in the task graph are modeled with state tables. Changes in the processing granularity (ROI processing), are modeled with a linear growth function. In Fig. 6, the effective latencies are shown for two data-partitioning strategies: serial and 2-stripe parallel.

For training the prediction models, we have used a data set of 37 video sequences of in total 1,921 video frames. In the training set, different scenarios exist to create the dynamics in algorithmic adaptation and switching. In Table 2(a), the Markov transition matrix is shown for the ridge-detection task. Similar matrices are generated for the couples selection and guide-wire extraction tasks. A summary of the prediction models can be found in Table 2(b). For the test sequences, an average prediction accuracy of 97% is reached with sporadic excursions of the prediction error up to 20-30%.

In Fig. 7(red curve, top), the computation time statistics are shown for a straightforward mapping. The latency at the output can vary between 60 and 120 ms. By feeding the computation time prediction from Triple-C to a runtime manager for semi-automatic
parallelization, we are already able to reduce the latency and jitter significantly. As our aim is to execute more functions on the same platform, also the memory and bandwidth predictions for different parallelization scenarios have to be taken into account in the future by the runtime manager.

Results show that our method is able to reduce the jitter on the latency significantly (Fig. 7, yellow curve, bottom). The throughput is much more stable compared to a straightforward mapping. However, there are still some small peaks noticeable, but the difference between worst-case execution and average-case execution is reduced to 20% in the semi-automatic parallel case, compared to a straightforward mapping leading to a latency variability of almost 85%.

8. Conclusions

In this paper, we have introduced a prediction model, called Triple-C, for Computation, Cache-memory and Communication-bandwidth usage for groups of dynamic image-processing tasks, employing scenario-based Markov chains. The models can be used to execute more functions on the same platform. It has been validated that the scenario-based Markov modeling is suited to describe the runtime resource usage, even if the flow graph dynamically switches between groups of tasks. As a consequence, it is possible to realize a parallelization of data distribution and computations, such that the latency is kept nearly constant. This feature enables the execution of more functions on the same platform.

Experimental results for a medical imaging function have shown that scenario-based modeling can be successfully applied to describe the resource usage function. Results show an average prediction accuracy of 97% with sporadic excursions of the prediction error up to 20-30%. Our semi-automatic parallelization strategy is able to lower the jitter on the latency with almost 70% and guarantees a constant throughput. The techniques described in this paper can potentially be used for alternative applications using image analysis, such as in surveillance systems.

References


Figure 7. Prediction model vs. actual computation time.

Authorized licensed use limited to: Eindhoven University of Technology. Downloaded on January 27, 2010 at 04:00 from IEEE Xplore. Restrictions apply.