Qos management of dynamic video tasks by task splitting and skipping

Citation for published version (APA):

DOI:
10.1109/ESTMED.2009.5336827

Document status and date:
Published: 01/01/2009

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
QoS Management of Dynamic Video Tasks by Task Splitting and Skipping

Rob Albers #*1, Eric Suijs *2, Peter H.N. de With #3

# Eindhoven University of Technology, Electrical Engineering
PO Box 513, 5600 MB Eindhoven, the Netherlands
1 r.albers@philips.com
3 p.h.n.de.with@tue.nl

* Philips Healthcare, Cardiovascular X-Ray R&D
PO Box 10.000, 5680 DA Best, the Netherlands
2 eric.suijs@philips.com

Abstract—We have integrated processing with deterministic and non-deterministic resource usage in an overall application and evaluated its performance on a multi-core processor platform. The non-determinism involves image analysis, which features a high variation in computing and memory requirements, as opposed to regular stream-oriented video processing. Quality-of-Service (QoS) control is based on resource-usage estimation functions. Scalability in parallel executing sub-functions is achieved by using task skipping and splitting as a concept, as every video application can be quickly made scalable in this way. The complete framework was validated for accurate latency control of an interactive medical imaging application. The proposed QoS mechanism runs fast enough to be executed in real time, and we achieved a reduction on the latency jitter of almost 70% for average-case processing, so that the quality can be significantly improved or an inexpensive hardware system can be employed.

I. INTRODUCTION

The design of video signal processing systems is gradually entering a new phase where instead of straightforward video processing for multimedia and quality improvements, the systems increasingly consider the actual content that is being processed and adapt the applied processing to that content. This ranges from extracting simple features such as color and texture, to an in-depth analysis of the video signal and its characteristic features. Examples of such applications are intelligent surveillance, face and behavior recognition, computer analysis of diseases in the medical domain, etc.

Also, computing platforms for executing these processing algorithms have been subject to continuous changes, but are now gradually converging towards multi-core architectures where depending on cost and application constraints, some of the cores are application-specific and others are generically programmable.

At first glance, video processing is well suited for multi-core processing, given the possibilities for dividing the tasks and the amounts of data that are involved in this type of processing, but the efficient mapping on the platform and the optimal distribution of tasks over the processor cores is a complex matter.

To be cost effective, it is required that the applications make efficient use of the available platform resources. The efficient mapping of multiple tasks onto a multi-core system is an unsolved case in the scientific literature. It is only recently that the first publications on this topic have become available, e.g. [1]. This paper attempts to contribute to the above problem exploration by concentrating on a smooth, efficient application control of both the computing resources and the bandwidth between the processing cores.

In more detail, the smooth control is required as the applications show considerable variations in the required computing effort, while our intention is to create a stable application performance, such as in latency or quality. This requirement has been explored for medical X-ray imaging, where we want to keep the output latency constant, to ensure sufficient response time for medical interaction, while still using a processing platform, dimensioned for average-case resource requirements.

II. CASE STUDY

Our case is based on medical imaging and analysis, involving advanced enhancement processing with strict latency constraints combined with image analysis and advanced conditional processing. The application solution space can be seen broader than medical imaging only, as many of the underlying processing tasks occur also in intelligent surveillance and alternative video content analysis applications such as face recognition, etc.

With minimal invasive interventions, cardiologists diagnose and treat coronary artery diseases using a catheter threaded through the arterial vessel tree to reach the heart. Coronary angioplasty is a catheter-based procedure to open up a blocked coronary artery and restore blood flow to the heart muscle. Analysis and motion-compensation techniques can improve the visualization and measurement of objects of interest (such as stents) in real-time X-ray imaging, thereby making it easier to perform optimal and complete stent placement. In this paper, we focus on a medical-imaging application to detect and enhance moving features, combined with a second application for increasing the image quality during an interventional angioplasty procedure [2].
In terms of video processing functions, image analysis and motion-compensation tasks are used to detect and enhance objects of interest. Whereas a first branch contains stream-oriented tasks having a deterministic nature in terms of computations and memory, a second branch contains data-dependent non-deterministic processing tasks, involving image analysis, which has a more dynamic nature than streaming video (Fig. 2). The dynamics come from the variable size of the image region to be analyzed, the initialization to find an object feature and the dynamic abortion of non-appropriate tasks when the detection or registration result is insufficient. In the flow graph of functions, the system first searches for important features, after which a motion-compensated registration (alignment) takes place. The result is an enhanced view of the features of interest (Fig. 1).

III. SYSTEM REQUIREMENTS

In the following, we describe a set of system aspects which are of key importance for the contents of this paper. These aspects have been organized in a bottom-up order, starting with the platform and ending with the applications executed on that platform.

- Multi-core platform - Motivated by the increasing gap between computing performance and memory bandwidth, the trend in processor design is inevitably in the direction of multiprocessor or multi-core systems. Depending on the nature of the applications, these systems may be generically programmable for each processor, or partially application-specific for often recurring tasks.

- Resource constraints and quality of service - In many applications and cases, cost constraints and/or performance constraints are imposed on the system. When optimizing the resource usage in combination with the best quality, applications should be controllable with respect to their resource requirements (and thus quality). Quality-of-Service (QoS) management tries to optimize the quality of individual applications, under the condition that a certain set of applications can still be executed in parallel and the overall system performance is optimal for the user.

- Application modeling with performance analysis - If the application tasks are well modeled and their required performance is known in the form of computation, memory and bandwidth requirements, the platform can be efficiently filled with a suitable set of applications. The key is that each application is modeled sufficiently accurately, so that a QoS manager can assign computing and memory resources to tasks accordingly. In this way, the overall performance can be optimized as well.

- Dynamic behavior of video/imaging applications - We have studied applications containing streaming-based, control-oriented and data-dependent analysis functions. These applications can be characterized by a significant amount of dynamic computing as the required amount strongly depends on the size of the objects which can also vary depending on the earlier obtained results. Furthermore, processing functions themselves can be data dependent, leading to considerable variations in execution time.

- Reconfigurability of processing tasks - When resource managers assign an increase or decrease on computing power or memory, the application has to be scaled accordingly. Since it is difficult to design a completely complexity-scalable application, we follow a more pragmatic approach and distinguish essential and non-essential processing tasks. Tasks can be scaled down or even skipped when they are non essential for the user. Essential tasks may be split and redistributed over the available processors.

IV. QUALITY CONTROL AND TASK SCALABILITY

A. Preliminaries

There will be system constraints on the available computing power, memory and bandwidth. If the computational load becomes too high for the platform, we intend to reduce...
the effort involved for particular tasks without the complete abortion of the task execution. In order to do this in a smooth efficient way, we need two elements:

- The applications should have scalable properties in terms of performance and computational effort.
- The platform should be able to control a number of parallel applications and their resource usage.

In the case study used throughout this paper, we discuss multiple objects and tasks running in parallel on a multi-core platform. This covers the cases of executing a single advanced application or a set of applications in parallel. This problem statement is a new element and distinguishes itself from previous work on QoS management [3][4]. We focus particularly on the application part of the overall resource management.

In order to allow single and multiple video applications in parallel to be controlled with the same QoS concept, we adopt the hierarchical QoS architecture from [5], where a Local QoS controls an individual application while a Global QoS controls the complete set of active applications and optimizes the system behavior. For our medical application, this means that the Global QoS controls the overall latency of the system, whereas the Local QoS optimizes the quality of individual applications under the overall latency constraint. The Local QoS can only optimize the quality of individual applications if they are scalable in quality and/or complexity. This complexity scalability is obtained by task splitting and skipping for essential and non-essential tasks.

### B. Hierarchical QoS Architecture

A hierarchical QoS system addresses both the optimization of resource allocation for individual applications and for a complete set of applications. Furthermore, we will not discuss the QoS service algorithms in high detail. Instead, we assume that an algorithm from literature can be adopted, given the broad availability of proposals on this topic [6][7]. In order to be able to provide reservation-based management, the resource management layer has to provide guaranteed services for allocation of resources. As the system allows a runtime adaptation of resources, also monitoring services are essential for the correct operation.

The QoS architecture has a hierarchical layered structure. It consists of two communicating resource managers, instead of the conventional single resource manager. The layered approach separates the system control optimizing overall quality and behavior from the responsibilities of individual application QoS units. The advantage of the layered approach is that the Local QoS control of individual applications can be designed along with the application and independent of the platform on which they will be executed.

Similarly, the Global QoS multi-core processor control can be designed without knowing the details of all applications that will be executed. Thus, applications can be reused on other platforms more easily. This separation of responsibilities supports compositionality and modularity of the system in order to add new applications to the existing system. It is inevitable that with two layers, a structure for communication between the two layers is required. The choice for a low-overhead communication protocol is preferred, as proposed by Pastnak in [5]. The responsibilities of the two QoS managers are as follows.

- **Global QoS manager** - It controls the total system performance involving all applications running in parallel. This manager optimizes the user benefits, such as the available parallelism and priorities of multiple applications, instead of only a single video application.
- **Local QoS manager** - It controls an individual application within the assigned resources, which were assigned by the Global QoS manager. This manager optimizes the application quality for the agreed amount of resources.

In Fig. 3, an overview of the architecture with QoS control is shown. Each application is divided into jobs and the platform supports the execution of each job. A job is defined as one full iteration of a task sequence within one image or a few images. The control of an application execution is conceptually visualized in Fig. 4. The differences between applications in terms of resource demands are hidden inside the Local QoS manager. The Local QoS units activates an internal resource estimator, that calculates the resource-usage requirements at all quality levels of the required jobs. This module has runtime requirements depending on the type of processing and input data characteristics.

The resource estimator is a job-specific component and depends on the implementation of the functional part of the job. The interested reader is referred to literature, where the authors describe resource-usage prediction techniques for image filtering [8] and image analysis [9].

Let us now specify the QoS problem in a more formal way. We describe the resource requirements of the job $i$ at
all defined quality settings (vector $q_i$) per resource $J$ by

$$R_{i,J}(q_i) = f_{i,J}(q_i).$$  \hspace{1cm} (1)$$

The resource type is $J \in \{C, D, I, B, T\}$, where $C$ denotes the computation resources per task, $D$ the data memory per task, $I$ the instruction memory per task, $B$ the required communication-port bandwidth, and $T$ the bandwidth on each connection between two tasks. The optimization problem for our Global QoS management can be formulated as

$$\max \sum_{i=0}^{N} \beta_i(q_i(c)),$$

with chosen quality $q_i(c)$, subject to

$$\sum_{i=1}^{N} R_{i,J}(q_i(c)) < \sum_{j=1}^{M} P_J(j),$$

with $J \in \{C, D, I, B, T\}$.

In the above equation, $\beta_i(q_i(c))$ denotes the benefit as the contribution of job $i$ to the user benefit, at selected quality level $c$, giving the quality $q_i(c)$. Parameter $N$ denotes the number of jobs and $M$ indicates the number of processing cores. Parameter $P_J$ is the total amount of a particular resource type $J$. The solution to the above problem statement is NP-hard and can therefore not be implemented at runtime. Instead, we apply the heuristic implementation from [10], which provides a near-optimal solution.

C. Task-level scalability

In this section, we change our attention from the QoS architecture to the application under study. Although advanced video and imaging applications are typically adaptive to many cases, they are not intrinsically developed to support QoS control. We have considered several options for implementing a smooth scalability in applications. The most attractive approach for this is using complexity scalability [11] [12], but it is not easy to design a full complexity-scalable application. For this reason, we concentrate on QoS control at the task level. Therefore, we have defined a new type of task scalability based on enabling, disabling, splitting and merging of tasks that compose a job. This form of scalability is much easier to implement and requires only knowledge of the algorithm at the task level. Based on the importance of the task processing, we distinguish essential tasks and non-essential tasks.

An application job is divided into communicating tasks. For each job and related quality level, we provide a detailed task graph, as portrayed by the general example in Fig. 2. The relevance of each task can be considered to contribute to the scalable QoS control of the underlying application. If a job contains idle tasks, then the corresponding communication resources are idle as well, which are denoted by dotted lines and gray circles (filter and feature detection tasks in Fig. 5). In the sequel, we experiment with task splitting and skipping to create a smooth scalable performance for the case study on professional medical imaging.

- **Scalability with task skipping** - We have implemented task skipping in the live-viewing stream (upper branch in Fig. 5) to illustrate task-level scalability for the non-essential tasks. The involved image enhancement tasks can be pushed to a lower quality level or even skipped to save (a part of the) assigned resources for other tasks, like the detection of features of interest in image analysis. In earlier work [8], we have presented a resource-usage model for streaming tasks. This model is useful when the mapping requires usage of different processor cores. For scalability, we have defined the following three quality levels Q of our experimental medical image enhancement application:

  - Q1: Basic quality and low resource demand; only contrast enhancement is applied.
  - Q2: High quality and resource demand; contrast enhancement and spatial filtering are applied.
  - Q3: Highest quality and resource-usage demand; the complete processing chain is executed.

- **Scalability with task splitting** - As task skipping can sometimes be too coarse in terms of impact on image quality or the impact on the release or fetching of platform resources, one can follow also a more selective approach, where the task may be split into sub tasks which can be distributed more easily over the processor cores. Besides this, it also solves another problem: skipping essential tasks is not acceptable, since it would violate the desired functionality or the quality too much.

With task-splitting scalability for essential tasks, the dynamics in the latency and throughput can be minimized by incidentally reconfiguring the computing tasks in the flow graph, i.e. freeing or consuming some of the resources that were prior budgeted for background tasks. Fig. 5(bottom branch) outlines the task-level scalable version of motion-compensated feature enhancement. In earlier work [9], we have presented a resource-usage model for the dynamic feature enhancement tasks. Scalability at the task level is achieved by splitting the group of feature detection tasks (RDG and MKX EXT) into sub-tasks, resulting in a lower latency.
To create scalability in the analysis tasks, we have defined the following three quality levels $Q$ of our experimental medical image analysis application.

- **Q1**: Low resource-usage demand; feature detection is executed sequentially.
- **Q2**: Medium resource-usage demand; feature detection is split into two sub-tasks.
- **Q3**: High resource-usage demand; tasks feature detection is split into four sub-tasks.

Summarizing, the above two concepts for task splitting and skipping provide an improved scalability with respect to QoS control and a more fine-grained distribution of tasks over a multi-core processor system.

Let us briefly reflect on an alternative way of benefitting from task skipping. In the above, the benefit was employed for downscaling the quality to free up platform resources. As an alternative, which is more relevant for low-cost systems, it is possible to add functionality to a compute constrained platform. In more detail for our case study, when image analysis is the main application of interest and not all resources are allocated, the live-viewing stream in the upper branch of Fig. 2 can be enabled as well, together with a quality-controlled execution of the enhancement tasks.

In the sequel of this paper, we report on experimenting with the task-skipping and splitting scalability to optimize latency and throughput within a professional medical imaging application.

V. EXPERIMENTS AND RESULTS

We have implemented the full two-branch signal processing flow-graph of Fig. 5 and evaluated its performance behavior on a multicore system [13]. The overall application involves motion-compensated feature enhancement and live viewing simultaneously within an X-ray medical imaging system. Our approach is to use the presented hierarchical QoS concept to dynamically switch between quality levels. The dynamics in the latency and throughput are minimized by incidentally reconfiguring the tasks in the flow graph, i.e. freeing or consuming some of the resources that were previously budgeted for background tasks.

In the case study, we process uncompressed image frames ($1024 \times 1024$ pixels, $30$ Hz), where live viewing operates at $30$ Hz, and image analysis at $15$ Hz. The maximum allowed latency is therefore $33$ ms for live viewing, and $66$ ms for image analysis. Instead of pure image quality control, the experiments concentrate on keeping the output latency constant by dynamically splitting the essential tasks into sub-tasks and reconfiguration of tasks to processor cores. At the same time, the live-viewing complexity and the corresponding image quality is decreased for short periods of time by skipping non-essential filtering tasks. The actual selection is based on the resource demand for the image analysis tasks, estimated by the prediction model in the local QoS controller.

As motion-compensated feature enhancement contains non-deterministic tasks, 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 **worst-case** solution is to employ a task partitioning on the platform, based on a worst-case resource reservation. Subsequently, at the end of the pipeline, a task with a variable delay keeps the latency and throughput constant. With this solution, the reserved resource budget is set too conservative for most of the time. Furthermore, the latency is much higher than actually required and it is impossible to exploit the difference between the average-case and worst-case requirements without affecting the guaranteed performance of the application.

For the (non-deterministic) image analysis application, the worst-case execution shows heavy excursions (85%) on the effective latency and the computation latency can vary between 60 and 120 ms (not shown in Fig. 6).

The previous observations and findings make it factually impossible to use dynamic applications in latency constrained applications, without over-dimensioning the system. Therefore, we have further experimented with our presented QoS concept for latency optimization. Using task splitting and skipping in our full application, the latency requirements can be met. The variation on the latency for image analysis (lower branch in Fig. 5) is reduced significantly to only 20% (see Fig. 6). For the live-viewing application running in parallel (upper branch in Fig. 5), the latency varies around $30$ ms. This variation results from the switching between different quality levels.

For image analysis, we perform task splitting at Q2 and Q3, where for Q2, the tasks are split in two, and for Q3 the tasks are split in four. For live viewing, the image quality is scaled in three quality levels, which means skipping (parts of) the filtering tasks.
VI. CONCLUSIONS

We have integrated image analysis processing with non-deterministic resource usage, and a regular stream-oriented video processing with deterministic resource usage in one overall application and evaluated its performance on a multicore processor platform.

We have used an available concept for hierarchical QoS control and employed this for a smooth and efficient mapping of the overall application onto the multi-core processor architecture. We have proposed and employed task skipping and splitting to create quality-scalable applications when applications share constrained platform resources. It was shown that this concept can be favorably used to ensure that latency requirements of non-deterministic applications are met, and even to add functionality when resources are still available.

The complete framework was validated for a case study on medical X-ray imaging, for which the jitter on the latency was reduced with almost 70%, compared to a worst-case mapping. The presented results in this paper are rather relevant for the near future, since dynamic video applications like image analysis are increasingly found in both the consumer and professional domain.

REFERENCES


