Exploiting Throughput for Pipeline Execution in Streaming Image Processing Applications*

F. Guirado[1], A. Ripoll[2], C. Roig[1], A. Hernàndez[3], and E. Luque[2]

[1] Universitat de Lleida
f.guirado@diei.udl.es, roig@diei.udl.es
[2] Univ. Autònoma of Barcelona
ana.ripoll@uab.es, emilio.luque@uab.es
[3] Computer Vision Center. Univ. Autònoma of Barcelona
aura@cvc.uab.es

Abstract. There is a large range of image processing applications that act on an input sequence of image frames that are continuously received. Throughput is a key performance measure to be optimized when executing them. In this paper we propose a new task replication methodology for optimizing throughput for an image processing application in the field of medicine. The results show that by applying the proposed methodology we are able to achieve the desired throughput in all cases, in such a way that the input frames can be processed at any given rate.

1 Introduction

There is a large range of emerging applications in which data generated in a given external environment is pushed asynchronously to servers that process this information. These applications are characterized by the need to process different instances of an input data stream in a timely and responsive fashion. Hereafter, we refer to such applications as streaming applications [1] [2] [3] [4].

There are two distinct criteria for judging the quality of an execution for these streaming applications: latency and throughput. Latency is the time taken to process individual data, while throughput is the aggregate rate at which the instances of the input data stream are processed. Throughput can also be measured in terms of its inverse, the Iteration Period (IP), which corresponds to the interval of time existing between the execution of two consecutive instances of data.

In this paper, we deal with image-processing applications executing in a streaming manner. These applications are typically composed of a set of computation stages performing different functions. The usual computations in the stages of these kinds of applications are blurring, filtering, interpolation, etc. As they are independent functions, they can be arranged in a set of consecutive stages that at a given time can be simultaneously processing different image

* This work was supported by the MEyC-Spain under contract TIN 2004-03388.

W.E. Nagel et al. (Eds.): Euro-Par 2006, LNCS 4128, pp. 1095–1105, 2006.

frames in pipeline fashion. The sequence of input frames is continuously received. Thus, throughput is a key performance measure to be optimized in these executions [2] [5].

In this context, we address the problem of maximizing the throughput of an image processing application in the field of medicine. The application under study is devoted to the detection of the real arterial structure from a sequence of Intra-Vascular Ultrasound (IVUS) images, captured by a transducer at a specific rate. Real-time constraints must be met in order for images to be processed at the same rate as that at which they are captured [6].

We first define the task model of the IVUS application by capturing its salient computational features. Based on this model, we propose a methodology that performs an innovative task replication technique for those tasks that can process independent input frames in such a way that a given throughput can be achieved. Then, the replicated application is executed in a simulation framework using a task mapping mechanism that considers its iterative behaviour.

We show through experimentation that the task replication technique allows us to reach the given throughput constraint in all cases. Additionally, we show the effectiveness of the whole strategy of replication and mapping in the optimization of processor utilization, as well as the speedup that is achieved.

The rest of the paper is organized as follows. Section 2 exposes the main characteristics and the steps that are performed in the IVUS imaging application. Section 3 describes the proposed methodology of task replication to exploit throughput. Section 4 outlines the main contributions of the literature in relation with the optimization problem that is undertaken in this paper. Section 5 shows the experimentation results that are obtained for the application under study. Finally, Section 6 outlines the main conclusions.

2 The IVUS Imaging Application

This is a study of an image processing application, in the field of medicine, for the detection of the real arterial structure (called adventitia) [7]. The input of the application is a sequence of IVUS frames that are captured by a radio-frequency transducer installed in a catheter. The captured data are sent out to be processed and then converted to images.

Tissue characterization is a fundamental tool for studying and diagnosing the pathologies and lesions associated to the vascular tree. This is an arduous job that requires specialists to manually identify the tissues and visualise them. IVUS imaging is a highly suitable visualization technique for the task as it provides a cross-section of the coronary vessel, revealing its histological properties and tissue organization [6].

As it is so time consuming and due to the subjectivity of the classification depending on the specialist, there is an increasing interest among the medical community in using automatic tissue characterization procedures. These automatic

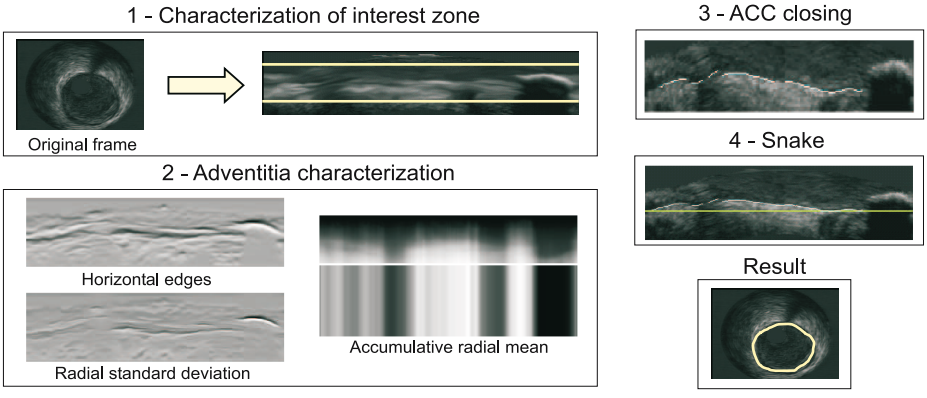


Fig. 1. Stages of the IVUS imaging application

procedures are time-critical, and consequently should provide answers in a minimum time. Figure 1 shows the main stages of the process that are explained below.

1. *Characterization of the interest zone.* Because the original image of adventitia is circular, it is transformed to polar coordinates. In this coordinate system, the adventitia appears as a dark horizontal line. By means of a diffusion method, the image is de-noised and the target structure is enhanced. Then, the band of interest is determined in order to reduce computational cost.
2. *Adventitia characterization.* The three following filters are applied to the image: horizontal edges, radial standard deviation and mean accumulative radial. The three filtered images provide the necessary information for discriminating between four different sets: adventitia, calcium, fibrous structures and the remaining pixels.
3. *Anisotropic contour closing (ACC).* The previous step characterizes the adventitia with a collection of fragmented curve segments. These segments are interpolated using ACC to join them.
4. *B-Snake.* Since the above interpolation process still presents gaps in side branches and calcium sectors, a parametric B-snake is used on the ACC closure in order to close it and obtain a compact and explicit representation. Finally, the identified adventitia is returned to cartesian coordinates to visualize its original circular shape.

3 The Optimization Problem

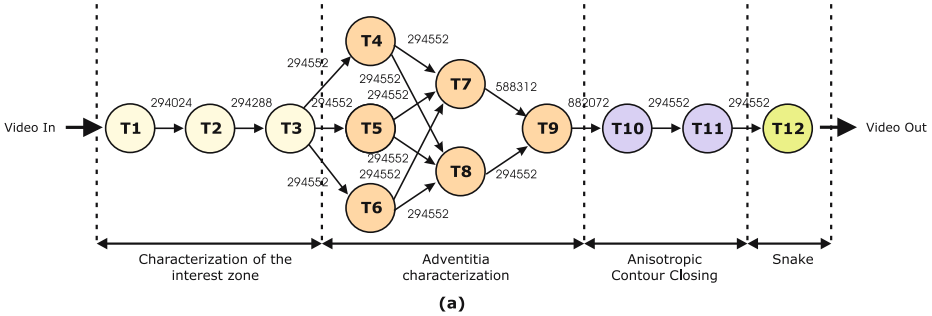
One of the key problems that arise when executing image-processing applications that act on an input sequence of image frames is having enough throughput to permit their processing at a given rate. We address the optimization of the throughput of these applications by considering the definition of computation stages that can run in pipeline.

The proposed methodology is based on two steps: in the first, the task graph model of the application is obtained. Based on this model, in the second step we apply the convenient replication of tasks that makes it possible to reach a desired throughput.

3.1 The Task Model

To exploit parallelism in streaming image processing applications, the main issue is to identify the sequence of different functions (steps) that are carried out for each input frame. Then, a parallel design of the application can be undertaken in such a way that the different functions are implemented as tasks that can run overlapped in pipeline for different image frames of the input stream.

The task model that we extracted for the IVUS imaging application is composed of 12 tasks that can be modeled using the directed acyclic graph (DAG) structure, G(V,E), illustrated in Figure 2(a). The graph is composed of a set of nodes V, each node representing a task. Each task Ti∈V has an associated computation time $\mu(Ti)$. E is the set of arcs representing task precedences. Each arc (Ti,Tj)∈E has an associated communication volume c(Ti,Tj), in bytes, to be transferred between tasks.



(a)

Task	Function	Time(sec.)
T1	Preprocessing	0,129
T2	CM calculation	0,366
T3	Difussion	1,713
T4	Edges calculation	0,013
T5	Varianza calculation	0,015
T6	CRM calculation	0,009
T7	Calcium Mask	0,029
T8	Adventitia Mask	0,003
T9	3D continuity filter/select	0,012
T10	Acc Closing	0,395
T11	Selection Cont.	0,052
T12	Snake	0,547

(b)

Fig. 2. (a) Task graph model of IVUS imaging application. (b) Functionality and computation time of each task.

Each step of the application can have several tasks performing different functions as indicated in the graph. Figure 2(b) shows the different functions that were identified and their correspondence with the tasks in the graph, together with the task execution time, in seconds. To obtain these task execution times we

profiled the execution of the different functions that conform a task in the IVUS sequential algorithm. We also computed the data structures that are shared between functions to determine the communication volume of data that has to be transferred between tasks. The sequential application was programmed in Matlab and executed on a Pentium IV processor at 3GHz with 512 Mb of RAM running windows.

3.2 Task Replication Method

In this subsection, we expose the methodology that is proposed in this paper to achieve a given throughput for streaming applications running in pipeline. We assume that the instances of the input stream have no temporal dependencies between themselves, as is the case with IVUS imaging. Thus, different image frames can be processed concurrently in the same step with replicated tasks. Consequently, throughput is improved, since multiple images are processed in parallel. We propose this methodology instead the typical data parallel approach at the application level, in order to exploit the task parallel capacity without penalizing latency.

Given an IP to be achieved, the replication problem consists of determining the tasks that should be replicated and the number of replications for each. It is established in the literature that the optimum IP in a pipeline execution is given by the maximum computation time $\mu(Ti)$, considering that communications are performed concurrently [4][8]. We enhance this model by taking into account the fact that on several platforms communications cannot be overlapped. Thus, the optimum IP is also influenced by the global amount of communications that are transferred from one task. With these considerations in mind, we propose a replication methodology that consists of the two following steps:

1. *Determine the tasks to be replicated*
   All the tasks are evaluated to decide whether they have to be replicated or not. Each specific task Ti∈V with a communication to Tj∈V, will be replicated if it has accomplished one of the following two conditions: (a) The computation time of Ti is greater than IP or, (b) the communication time to transfer volume c(Ti,Tj) is greater than IP. Taking these conditions for replication into account, the algorithm shown in Figure 4 determines the tasks to be replicated, joined in subgraphs (replicable subgraphs). Starting from the DAG graph of the application, for each task Ti∈V it calls the recursive procedure *group*(Ti), which returns the set of successor-replicable tasks that form the replicable subgraph to which Ti belongs.

   From the identified subgraph, it ascertains whether there is an intersection with the previously found subgraphs that are stored in *subgraph_set*. Should the subgraph have a task in common with another, these are joined into a single subgraph. Then, the subgraph of replicable tasks is the chosen entity to which replication will be applied in the next step.
2. *Calculate the number of replications*
   In this step the number of copies is calculated for each replicable subgraph and the number of copies for each task inside the subgraph. This provides

us with the most appropriate number of replications for each task, instead of replicating the same number of copies for all the tasks, which, in some cases, would be an excessive replication.

To illustrate this step consider the example of Figure 3(a) where we can see a replicable subgraph with tasks Ti and Tj, along with 2 and 4 replications respectively. Figure 3(b) shows the result that would be obtained if the tasks were replicated individually, which leads to a large amount of dependencies. This could lead to an excessive overhead when communication from the same tasks is serialized as we have considered. Figure 3(c) illustrates the replication result applied on the subgraph level as proposed.

Figure 5 shows the algorithm to be applied to each replicable subgraph, which proceeds as follows. For each task Ti in the subgraph, the corresponding number of replications is calculated as the maximum between computation and communication, divided by the given IP. To calculate the number of replications of the graph we identify its initial tasks. Among these initial tasks, we chose the one with the lowest number of replications assigned. This determines the number of the replications of the whole subgraph.

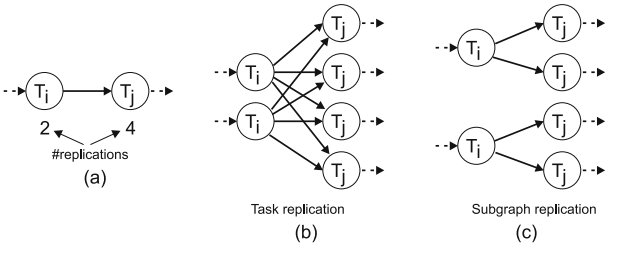


Fig. 3. (a) Replicable subgraph. (b) Result of replication if it was applied on a task level. (c) Result of replication with our method.

```
subgraph_set=∅
non_evaluated_tasks={Ti; Ti∈V}
for each Ti ∈ non_evaluated_tasks
 subgraph=group(Ti)
 if subgraph ≠ ∅
    for each H ∈ subgraph_set
      if H ⋂ subgraph ≠ ∅ then
        subgraph = subgraph ⋃ H
    end_for
    subgraph_set=
    =subgraph_set ⋃ subgraph
 end_if
end_for
```

```
function group(Ti)
 adjacent_set=∅
 non_evaluated_tasks=non_evaluated_tasks-{Ti}
 if is_replicable(Ti)
   adjacent_set={Ti}
   for each task successor task Tj ∈ V and
   is_replicable(Tj)
     adjacent_set=adjacent_set ⋃ group(Tj)
   end_for
 end_if
 return adjacent_set
end_function group
```

Fig. 4. Algorithm for identifying replicable subgraphs of tasks

```
function replication
  for each subgraph ∈ subgraph_set
    for each Ti ∈ subgraph
        number_replications[Ti]=⌈ max(μ(Ti),max(∀Tj successor_of Ti comm(Ti→Tj))) / iteration_period ⌉
    end_for
    Tinit = initial task of subgraph with lowest number_replications[Ti]
    number_replications_subgraph = number_replications[Tinit]
    for each Ti ∈ subgraph
        number_replications[Ti]= ⌈ number_replications[Ti] / number_replications[Tinit] ⌉
    end_for
  end_for
end_function replication
```

Fig. 5. Algorithm for determining the number of replications of the subgraphs and their internal tasks

Table 1 shows the development of the replication method when it is applied to the task graph of the IVUS application for a desired throughput of 214 ms. In this case, we identify three replicable subgraphs (two with a single task). Figure 6 graphically shows the result of this replication method in IVUS application.

Table 1. Development of the replication method for IVUS application

Rep.subgraphs	n_rep.[Ti]	n_rep. inside subgraph	n_rep_subgraph
T2 → T3	T2:2	2/2=1	2
	T3:8	8/2=4	
T10	T10:2	2/2=1	2
T12	T12:3	3/3=1	3

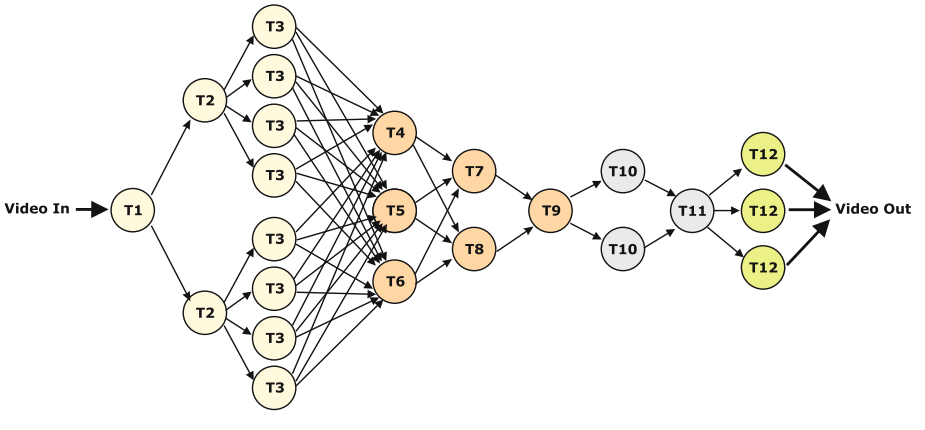


Fig. 6. Ivus replicated graph

4 Related Work

The optimization of throughput in streaming applications running in pipeline has been undertaken by several proposals in the literature. Some authors provide generic solutions to exploit throughput under different constraints without considering replication. Hoang and Rabey in [2] propose an algorithm that solves a resource-optimization problem and maximizes throughput. Hoang and Rabey's proposal was later improved by Yang et al in [3]. This starts the task assignment with the ETF algorithm that is based on a classic DAG heuristic [9]. From the obtained assignment, it processed additional reassignment steps in order to exploit the iterative behaviour of applications. In all these approaches, the maximum throughput is given by the maximum computation time of the tasks in the application, which also indicates the minimum IP achievable.

In order to improve throughput there are approaches that introduce the concept of replication to their techniques, dividing the input frame into several parts and applying data parallelism to each [4] [8]. Lee et al in [5] apply a replication technique where the tasks of the same stage have to be identical. Unlike these previous works, our approach considers the possibility of replication on the task level for applications with arbitrary task structure and without constraints on the kind of tasks within each stage. The replicated tasks perform the same computation for different frames of the input stream. This facilitates the applicability of the replication method as the code of the task does not need to be modified and provides a more feasible solution for improving throughput.

5 Experimentation Results

In this section, we conducted an experiment to show both the applicability and the benefits provided by the proposed approach of task replication when used to execute the IVUS imaging application in a cluster environment.

From the obtained IVUS task graph model, we executed the application in the simulation framework pMAP [10], which simulates the execution of message-passing applications in distributed systems. The underlying system was modelled by defining a set of homogeneous nodes with the same characteristics as those used in the sequential execution. The network was modelled as a Gigabit Ethernet.

The replication method was evaluated for a desired IP that is based on $\mu(T3)$, which is the highest computation time in the graph and consequently indicates the maximum throughput that is achievable. Thus, we replicated the application tasks using our methodology in order to increase throughput by 2, 3, 4, 8 and 16 times, indicated as x2, x3, x4, x8 and x16 respectively. The tasks of the replicated graph were assigned to the processors using a specific mapping mechanism for pipeline applications [11] that makes it possible to exploit throughput and optimize the number of processors.

Figure 7 shows the throughput, in frames/second, that was obtained for the application using our methodology, compared with the maximum throughput

that can be theoretically obtained for the different number of replications. As can be observed, we obtained significant similarities for both values. Thus, the replication method is able to achieve the given throughput constraints for the IVUS imaging application.

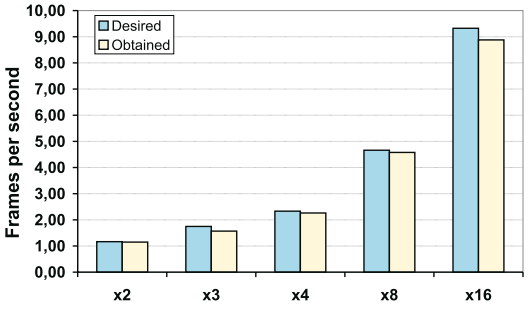


Fig. 7. Throughput

As the number of processors that are used is increased due to task replication, we evaluated the processor utilization in the executions. For each experiment, Figure 8(a) shows the number of required processors with the corresponding average utilization. As can be observed, the worst case is an average utilization of 68% when 4 replications were applied. In all the remaining cases, the percentage utilization is greater that 70%. Thus, the mapping mechanism applied after replication is able to optimize resource utilization.

Finally, we evaluated the speedup in order to analyse the influence of the increase in the number of tasks and processors due to replications. As shown in Figure 8(b), the obtained speedup has the same tendency as the maximum, and the difference between both becomes more significant only when 36 processors were used in the x16 experiment.

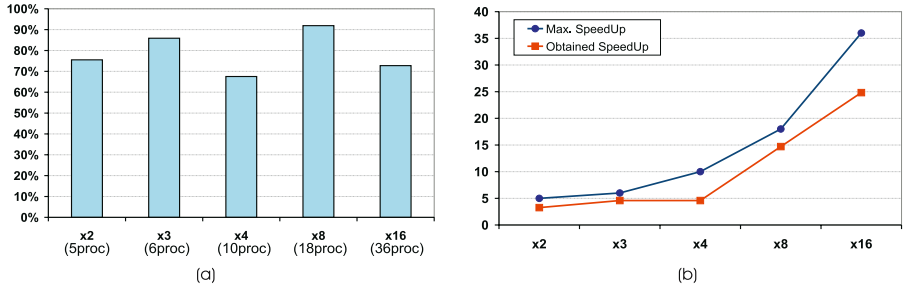


Fig. 8. (a) CPU utilization. (b) Speedup.

6 Conclusions

Optimization of throughput is a key performance measure for optimization in image processing applications that act on an input sequence of image frames. In this work, we addressed the exploitation of throughput based on a real application, IVUS imaging, in the field of medicine.

We have proposed a task replication methodology that consists of two steps: (a) obtaining the task graph model of the application and, (b) applying the convenient replication of tasks to enable the desired throughput.

The effectiveness of the proposed approach was evaluated for the IVUS imaging application. For different values of throughput to be obtained, we applied replication, and the replicated application was executed through simulation in a cluster environment using a task mapping mechanism that considers its iterative behaviour. The results show that the proposed replication method followed by the mapping mechanism is able to achieve the desired throughput for the application under study while maintaining good utilization of resources.

References

1. M. Lee, W. Liu and V. K. Prasanna: A Mapping Methodology for Designing Software Task Pipelines for Embedded Signal Processing. Proc. Workshop on Embedded HPC Systems and Applications of IPPS/SPDP 1998. Pp. 937-944.
2. P. Hoang and J. Rabey: Scheduling of DSP Programs onto Multiprocessors for Maximum Throughput. IEEE Trans. Signal Processing. Vol. 41. N. 6. Pp. 2225-2235. June 1993.
3. M-T. Yang, R. Kasturi and A. Sivasubramaniam: A Pipeline-Based Approach for Scheduling Video Processing Algorithms on NOW. IEEE Trans. on Par. and Distr. Systems. Vol. 14. N. 2. Pp. 119-130. Feb. 2003.
4. A. Choudhary, W. K. Liao, D. Weiner, P. Varshwey, R. Linderman and M. Linderman: Design Implementation and Evaluation of Parallel Pipelined STAP on Parallel Computers. Proc. 12th Int. Parallel Processing Symposium. Florida. Pp. 220-225. April 1998.
5. M. Lee, W. Liu and V. K. Prasanna: Parallel Implementation of a Class of Adaptive Signal Processing Applications. Algorithmica. N. 30. Pp. 645-684. 2001.
6. O. Pujol and P. Radeva: Supervised texture classification for intravascular tissue characterization. Handbook of Medical Imaging. Vol. 2. Pp. 57-110. Kluwer Academic/Plenum Pub. 2005.
7. D. Gil, A. Hernàndez, O. Rodríguez, J. Mauri, P. Radeva: Statistical Strategy for Anisotropic Adventitia Modelling in IVUS. IEEE Trans. on Medical Imaging. (in press)
8. Subhlok J. and Vongran G.: Optimal Use of Mixed Task and Data Parallelism for Pipelined Computations. J. Par. Distr. Computing. Vol. 60. Pp 297-319. 2000.
9. J.J. Hwang, Y. C. Chow, F. D. Anger and C. Y. Lee: Scheduling Precedence Task Graphs in Systems with Interprocessor Communication Times. SIAM J. Comp. Vol. 18. N. 2. Pp. 244-257. April 1989.

10. F. Guirado, A. Ripoll, C. Roig and E. Luque: Performance Prediction Using an Application Oriented Mapping Tool. IEEE Proc. Euromicro Conf. on Parallel, Distributed and Network-based Processing. (PDP) Pp. 184-191. Feb. 2004.
11. F. Guirado, A. Ripoll, C. Roig and E. Luque: Exploitation of Parallelism for Applications with an Input Data Stream: Optimal Resource-Throughput Tradeoffs. IEEE Proc. Euromicro Conf. on Parallel, Distributed and Network-based Processing. (PDP) Pp. 170-178. Feb. 2005.