User Transparent Parallel Image Processing

Seinstra, F.J.

Publication date
2003

Chapter 8

Summary and Discussion

"From a word to a word I was led to a word,
From a deed to another deed."

Excerpt from *The Poetic Edda* (Iceland, ca. 1280)

8.1 Summary

To satisfy the performance requirements of current and future applications in image-, video-, and multimedia processing, the image processing community at large exhibits an overwhelming desire to employ the speed potential of high performance computer architectures. Unfortunately, there is a major discrepancy between the need for *easily obtainable* speed in imaging, and the potential of current high performance computers to fulfill this need. Primarily, we ascribe this problem to the fact that no programming tool is available that serves as an effective aid in the development of image processing applications for parallel and distributed systems. Existing tools generally require the user to have a thorough insight in the complexities of parallelization, often at a level of detail far beyond that of non-dedicated parallel programmers. As it is unrealistic to expect researchers in imaging to become experts in high performance computing, it is essential to provide a tool that shields its users from all intricacies of parallelization.

The work described in this thesis is an endeavor to bridge the gap between the expertise of researchers in image processing, and the particular skills required for efficient employment of high performance hardware architectures. To this end, we describe the design and implementation of an innovative software architecture that allows its users to develop parallel image processing applications in a *user transparent* (i.e., fully sequential) manner. We explore the requirements such architecture must adhere to for it to serve as a long-term solution for the image processing community. Also, we provide a detailed discussion of each of the architecture's constituent components, and the research issues associated with each of these. Finally, we evaluate the provided performance gains, to see how these compare to reasonable hand-coded applications.

In Chapter 2, we investigate the applicability of existing high performance hardware architectures and associated parallelization tools in the field of image processing. Based on a set of requirements we conclude that homogeneous Beowulf-type commodity clusters constitute the most appropriate class of target platforms — most importantly due to the emphasis on price-performance. The evaluation of associated software tools shows library-based environments to offer a solution that is most likely to be acceptable to the image processing community. Primarily, this is because these are most easily provided with a programming model that offers full *user transparency*. However, due to insufficient *sustainability* levels, no user transparent tool is found to provide an acceptable *long-term* solution. Based on these observations we propose a new library-based software architecture for parallel image processing on homogeneous Beowulf-type commodity clusters. Due to its innovative design and implementation the architecture fully adheres to the requirements of user transparency and long term sustainability. Consequently, the architecture constitutes a solution that is likely to be acceptable as a long-term solution for the image processing community at large.

In Chapter 3, we present the design philosophy behind the parallel image processing library, which is the core component of the developed software architecture. Primarily, we focus on the problem of implementing the library such that code redundancy is avoided as much as possible, whilst ensuring efficiency of parallel execution. To this end, we introduce the notion of *parallelizable patterns*, and discuss how parallel implementations are easily obtained by sequential concatenation of operations that are *separately available* in the library. More specifically, on the basis of a set of four *data access pattern types*, we define a default parallelization strategy for any operation that maps onto one of two parallelizable pattern types. For each parallel operation this default strategy is optimal, as it fully exploits the available parallelism with minimal communication overhead. As such, we demonstrate that the presented design philosophy allows for long-term architecture sustainability, as well as high efficiency.

In Chapter 4, we indicate how to apply a simple analytical performance model in the process of automatic parallelization and optimization of complete image processing applications. Existing approaches generally incorporate a direct relationship between the estimation accuracy and the model's complexity (and thus: efficiency of evaluation). To deal with this problem, we propose a *semi-empirical modeling* technique. While being simple and portable, the approach also provides a sufficiently high estimation accuracy. The approach is based on a high level abstract parallel image processing machine (or *APIPM*) definition, which is designed to capture typical run time behavior of parallel low level image operations. From the related APIPM instruction set, a high level model is obtained that is applicable to all machines in the class of target platforms. The essence of the semi-empirical modeling approach is that any behavior or cost factor that can not be assumed identical for all target platforms (such as interprocess communication, or caching) is abstracted from in the definition of the model parameters. To still bind each abstract model parameter to an accurate performance estimation for a parallel machine at hand, benchmarking is performed on a small set of sample data to capture all such essential, but implicit cost factors. A comparison of model estimations and experimental measurements indicates that, for realistic applications, the APIPM-based performance models are highly accurate.

Chapter 5 extends the APIPM-based performance models for more accurate estimation of the MPI message passing primitives used in the library implementations. Existing communication models (such as LogP) do not incorporate all capabilities of MPI's send and receive operations. So far, the (often significant) effect of memory layout on communication costs has been ignored completely. In our software architecture, a higher predictive power is essential to perform the important task of automatic and optimal distribution of image data structures. To this end, we define a new model (called *P-3PC*), that closely matches the behavior of MPI's standard point-to-point operations. First, the model accounts for differences in performance at the sender, the receiver, and the full communication path. Also, it models the impact of memory layout, and accounts for communication costs that are not linearly dependent on message size. Experiments performed on two significantly different cluster architectures indicate that, in comparison with related models, P-3PC is capable of more accurate estimation of the communication overhead of typical image processing applications.

Chapter 6 discusses the automatic conversion of any sequential image processing application into a legal, correct, and efficient parallel version. To this end, we define a finite state machine (fsm) specification that guarantees the process to be performed correctly at all times. First, the fsm is shown to bring about a surprisingly simple and efficient approach (called *lazy parallelization*) for communication cost minimization. For further optimization, the fsm is used in the construction of an application state transition graph (*ASTG*), that characterizes an application's run time behavior, and also incorporates all possible (combinations of) parallelization and optimization decisions. As each decision is annotated with a run time cost estimation obtained from the APIPM-based performance models, the fastest version of the program is represented by the cheapest branch in the ASTG. As the issue of automatic optimization of complete applications is the central, most essential problem our software architecture for user transparent parallel image processing is confronted with, the applied solution combines all of the results obtained in Chapters 3, 4, and 5.

In Chapter 7, we give an assessment of our architecture's effectiveness in providing significant performance gains. We describe the implementation and automatic parallelization of three well-known example applications that contain many operations commonly applied in image processing research. From the evaluation, we conclude that the performance obtained with the parallel versions generated by our software architecture compares well to that of reasonable hand-coded parallel implementations.

8.2 Discussion

In this thesis we have aimed at the development of an effective programming tool that provides sustainable support in the implementation of parallel image processing software by non-experts in high-performance computing. We believe that we have succeeded in that mission. In the very first place because the architecture shields the user from *all* parallelization and optimization issues. As such, the architecture can be used immediately, without requiring additional knowledge from the application programmer. In the second place because the architecture allows its developers to respond to changing demands and environments quickly and elegantly. The applied design philosophy has largely inherited this property from the sequential image library (Horus) the architecture is based on (see Chapter 3). Finally, because the obtained efficiency was shown to be comparable to that of reasonable hand-optimized code. This implies that the parallelization overhead induced by the architecture is marginal. For these reasons we conclude that the software architecture fully adheres to the requirements of user transparency and sustainability as put forward in Section 2.3.1.

Despite this result, certain properties of the software architecture as described in this thesis are not always desirable. A first problem is due to the extensive use of abstractions incorporated in the architecture. As pointed out in Chapter 3, among the advantages of the use of abstraction and (parallelizable) patterns are a huge reduction in human software engineering effort, and enhanced software maintainability, extensibility, reusability, and portability. However, there is a trade-off between the use of specific and abstract libraries in terms of (sequential) processing speed. Also, abstract libraries may have a long compilation time and large footprints due to the automatic expansion of function instantiations. One way to overcome these disadvantages is to build a tool that can automatically generate specific (even tailor-made) image processing libraries from the existing abstract implementations.

A second issue that was not discussed in this thesis, is the question of how to deal with enormous amounts of input data. Applications working on video-sequences, or complete image databases, may suffer from a significant I/O performance bottleneck. Therefore, it is essential to re-evaluate data I/O in our architecture, and to incorporate optimizations accordingly. One solution may be to use data compression techniques in software. However, research related to this issue may also lead to the conclusion that hardware extensions (e.g., MPEG-encoders and -decoders) are essential.

A problem with *any* library-based environment is that it can never provide a *complete coverage* of all desired functionality. As stated in Chapter 3, we estimate that the algorithmic patterns available in our library cover over 90% of all low level imaging operations. Additional patterns, such as for recursive neighborhood operations, and queue-based algorithms are currently under construction. Other algorithms (a.o., data dependent operation) may not map onto one of the standard patterns, and also may not parallelize well, because of irregular data access. In spite of this, we do not expect a large amount of patterns to be added still, as one can compute only a limited variety.

In conclusion: the work described in this thesis indicates that it is possible to provide an effective long-term solution to a difficult problem, basically by incorporating a set of relevant high-level abstractions, and applying relatively simple methods for resolution of each constituent sub-problem. We believe that a similar approach could be applicable in other fields of research as well, especially in areas where the set of typical operations is limited — as is the case in low level image processing. However, only time will tell whether our software architecture for user transparent parallel image processing indeed is the effective long-term solution we consider it to be.