

Comparative Analysis of Design Alternatives in Embedded Systems*

James E. Hilger¹ Insup Lee,² Oleg Sokolsky²

¹ US Army CECOM Night Vision & Electronic Sensors Directorate

² Department of Computer Science, University of Pennsylvania

June 9, 2000

Abstract

The paper addresses the problem of analysis of alternatives in the design of distributed real-time systems. In the design of such a system, a hardware architecture needs to be chosen; then the system algorithm must be mapped onto the chosen architecture. As a result, the design space is very large, and evaluation of alternatives is very expensive. We propose an approach to an approximate analysis of alternative design decisions, which allows to eliminate infeasible solutions faster.

1 Introduction

Following the advances in processing and sensor technologies, more and more powerful tasks can be implemented by embedded real-time systems. Operational requirements, especially timing constraints, for such systems become ever more stringent. Parallel and distributed processing has to be used in order to meet these requirements.

Because of this, design and implementation of embedded systems is a very challenging task. With multiple processors in a system, the number of alternative implementations of an algorithm that have to be considered by designers increases dramatically. On the one hand, a hardware architecture has to be selected along with a target platform. On the other hand, the algorithm has to be mapped onto the chosen architecture, which can be done in several ways. All of these choices can have dramatic

impact on the system performance, which may be hard to estimate in advance. It would be, of course, prohibitively expensive to implement several alternatives and evaluate them later. System analysis techniques have to be used in order to estimate performance of each alternative.

The most commonly used technique currently in use is simulation [2]. Simulators capable of very detailed modeling of an algorithm mapped on a distributed system are available, and allow designers to achieve highly accurate estimates. Unfortunately, detailed simulation is a very lengthy process. It would be infeasible, timewise, to analyze all design alternatives by simulation. Complete simulation of just one alternative can take months to complete. Therefore, there is a need for an alternative analysis technique that would yield results fast, even if these results would have less precision than simulation.

This paper proposes, as an alternative to simulation, an analysis methodology based on formal methods. Formal methods, a collection of specification and analysis techniques based on a mathematical description of a system, provide a rigorous way of exploring behavior of a system specification. Numerous tools are available to help the user in this exploration.

In the proposed approach, an algorithm is partitioned into a set of tasks. Each task can be run on a single processor, competing for processing time with other tasks assigned to the same processor. The user specifies the amount of processing time necessary for the completion of the task, and the interconnection between tasks - that is, the size of data elements exchanged between tasks and the fre-

*This research was supported in part by NSF CCR-9619910, ARO DAAG55-98-1-0393, ARO DAAG55-98-1-0466

quency of exchanges. Data is exchanged between processors along communication channels. Several tasks may have to share a channel.

Based on this description, a formal specification of the system is constructed and analyzed by a formal methods tool. The specification is formed by a number of building blocks, each representing a system component: a processor running a certain task or set of tasks, a communication channel shared by a set of tasks, a queue for temporary storage of data, etc. Each of these components is represented by a parameterized specification, which can be instantiated according to the description of the algorithm. Since we are interested in an approximation of the system behavior, the specifications do not have to be detailed and their analysis can be performed fast, compared to simulation.

As a case study of the possibilities of this methodology, we performed analysis of an automatic target recognition (ATR) system for an Army ground vehicle. The tool used for analysis is PARAGON [3], a toolset for formal specification and analysis of real-time systems.

2 The Problem

Many mappings of an algorithm onto a multiprocessor system are usually possible. Simulation is traditionally used to evaluate different mappings and choose those that meet timing requirements for the system. However, simulation of a multiprocessor system is a laborious task, and the number of alternatives precludes us from evaluating all of them within the design cycle. Performance differences between implementations resulting from different mappings can be very significant. It is therefore important to identify those mappings that may yield feasible solutions early in the design cycle.

We approach this challenging problem by means of high-level modeling of the embedded system. Models of the system implementations are constructed and analyzed for their parameters, such as the number of processors necessary to meet the timing requirements. This analysis yields very approximate results, which nevertheless allow us to identify and discard infeasible solutions early in the design cycle.

The algorithm of the embedded system is modeled as a collection of tasks, each task assigned to

a certain processor. A task corresponds to a step of the algorithm applied to a fraction of the input data. A task interacts with other tasks by sending results of its computation to the processor responsible for the subsequent round. Interaction between tasks is performed by communication channels on the processor board, or by inter-board channels. These channels are modeled to reflect their latency and transmission times. In addition, data transmitted between tasks has to be buffered to allow each task to proceed at its own speed.

The specification of an embedded system is constructed from a set of building blocks that represent commonly occurring system components. When modeled in this way, an embedded system is represented as a parallel composition of a number of similar processes. We were able to categorize the several kinds of processes involved in this kind of a specification. As mentioned above, commonly used kinds of processes are (1) computation tasks assigned to a processor, that consume inputs from one channel and produce outputs on another channel; (2) communication channels that transmit data between processors; (3) shared channels that can carry data from multiple sources to multiple destinations; (4) queues or buffers for temporary storage of data.

For each of these types of data, we defined a parameterized process template that can be easily reused in many specifications by modifying its parameters. Parameters of a template depend on the kind of a process it represents. For example, computation tasks are parameterized by the duration of the task and the processor resource that runs the task; communication channels are parameterized by their latency and bandwidth; buffers have their capacity as parameters.

This categorization of the specification components allowed us to try different system configurations quickly. Indeed, to construct a specification of a new system configuration, we just need to instantiate appropriate process templates and describe their interconnections. The process templates are organized in a library, and an intuitive user interface for instantiating and connecting templates can be provided. This turns a general-purpose formal analysis tool into a framework for analysis of embedded system designs.

3 Application

3.1 System description

We tested the described methodology in a case study involving an implementation of a specific embedded system. The algorithm used in this study is an Army Research Laboratory (ARL) sponsored ATR algorithm named the ATR Relational Template Matching (ARTM) algorithm [1]. At a high level, the ARTM algorithm operates on input images looking for edges and boundaries which, when combined, have the shapes of targets. Initial target templates are used to look for gross shapes to separate target-like regions from clutter regions. As the algorithm progresses, the target templates used are refined to separate target classes and eventually individual targets. The ARTM algorithm consists of six stages, referred to as Rounds. Rounds 0 and 1 are for screening potential target pixels from background clutter pixels (target screening). Rounds 2 and 3 perform operations on pixels that pass Rounds 0 and 1. Rounds 2 and 3 are designed to converge on a specific target pose (target separation) while Round 4 verifies the target pose (target verification). The last Round performs proximity completion, i.e. a nonlinear form of spatial integration, and outputs a final target designation list for the input image. Algorithm control flow from Round 0 to Round 4 proceeds irregularly along a conditionally branching target hypothesis tree for each pixel in the input image. Control passes to the next Round only if the latest Rounds' results exceed the threshold, else it terminates. If control terminates, the pixel under test is no longer considered as a potential target pixel. Proper threshold design permits each Round to operate on progressively fewer pixels. The image pixels included in all of these computations vary in location depending on the particular target template. The total number of operations necessary to process a single 1315 pixel by 480 pixel image according to the ARTM algorithm is approximately 8,500 million. It should be noted that this operation count includes each addition necessary to address a particular pixel location as well as operations necessary to perform mathematical computations.

The operational scenario sets overall requirements such as frequency of results, which input data, and the ultimate format of the output data.

The primary function of the ATR in this case is to cue the human operator as to the location and presence of potential targets. This aids the human operator in the fatiguing task of manual search and detection. This is also necessary to meet the short timelines of target detection. It is the responsibility of the human operator to identify and prioritize targets among multiple target cues. For this study, the input data are digital InfraRed (IR) images that have 480 rows of 1315 pixels with each pixel represented by 12 bits. The input frame rate is 10 frames per second. This translates into a latency of 100 ms which represents the total time allotted to process the frame according to the algorithm.

In order to satisfy the stringent timing requirements, an implementation of the ATR system must be distributed across several processing components. The target multiprocessor system of this case study is based on 6U VME boards by Mercury Computing Systems, each containing four 400MHz PowerPC 750 processors. All integer instructions are taking 1 clock cycle, except multiplication (5 clock cycles) and division (19 cycles). Memory writes take 2 cycles. For this series of experiments, it is assumed that the superscalar integer unit of the processor allows it to execute, on the average, 1.5 integer instructions per clock cycle. The impact of cache misses degrades performance by 1.5. The bandwidth of the communication channels on the board is 160 Mbytes/s.

3.2 Analysis

The specification of the ATR system was constructed using the specification and verification toolset PARAGON [3]. PARAGON provides the means to write formal specifications and explore their behaviors. Analysis of the ATR algorithm was undertaken for several system configurations, featuring one, two and three processor boards. A sample two-board configuration is shown in Figure 1. One processor is responsible for distributing pixel data to processors on both boards. Five processors (three on board 1 and two on board 2) handle Round 0, one processor is dedicated to Round1, and all remaining rounds are processed on the last processor. While transmissions of data between processors on the same board can be carried out concurrently, inter-board transmissions all go through the same channel and are serialized. For simplicity,

all transmissions between the boards are assumed to take the same amount of time. described below.

The same hardware architecture has been analyzed with a different mapping of the tasks to processors: six processors were assigned to Round 0, while Round 1 was bundled on the same processor with all other rounds.

3.3 Analysis results

Each configuration has been applied to images of different sizes and analyzed to determine the frame rate that can be achieved by the configuration. Results of the experiments are summarized in Figure 2, showing the frame rate as a function of the number of processors dedicated to Round 0.

4 Conclusions and Future Work

We presented an approach to a rapid approximate analysis of alternatives in the design of safety-critical real-time systems. The goal of analysis is to obtain estimates of the computational resources needed to implement the system, and to eliminate infeasible alternatives. Alternatives that survive this approximate analysis are then evaluated in more detail using more precise - and more expensive - techniques. Preliminary results obtained through the case study suggest that this approach will be helpful in design of large safety-critical embedded systems.

Our future research on this topic will pursue several directions in order to make this approach more helpful and easy to use.

- An intuitive front-end to the analysis tool will be designed, to shield the user as much as possible from the low-level details of the specification and analysis techniques.
- The set of process templates will be enlarged and refined. The goal of this is twofold: on the one hand, new types of templates are needed to enlarge the scope of this technique; on the other hand, templates need to be made more flexible to allow different degrees of precision during analysis.

- Additional case studies need to be performed to gain more experience, which will allow us to improve the technique further.

References

- [1] T. Kipp *et al.* ATR Relational Template Matching, Phase 1 Final S & T, Report, Army Research Laboratory, Alliant Techsystems, Inc., and Mathematical Technologies, Inc. Contract DAAB07-90-C-F427, 1990.
- [2] J. E. Hilger. Adaptive computing technology: An enabling processing technology for advanced sensor systems. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, October 1998.
- [3] O. Sokolsky, I. Lee, and H. Ben-Abdallah. Specification and analysis of real-time systems with PARAGON. *Annals of Software Engineering*, 1999. Accepted for publication.

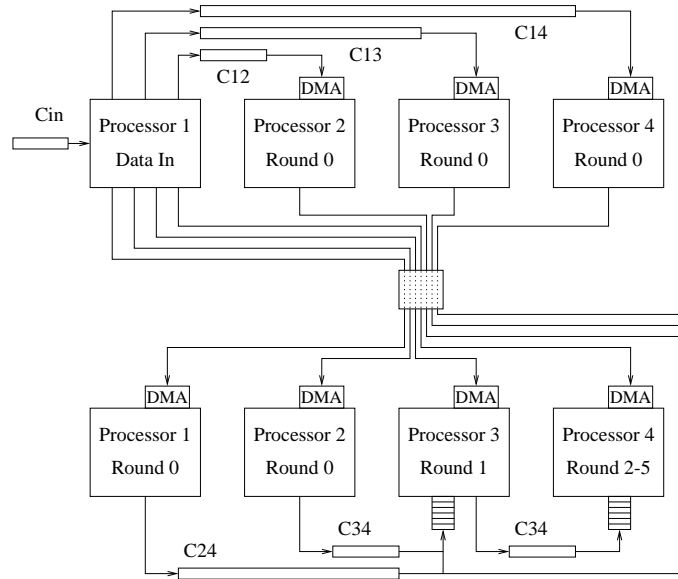


Figure 1: Sample ATR architecture

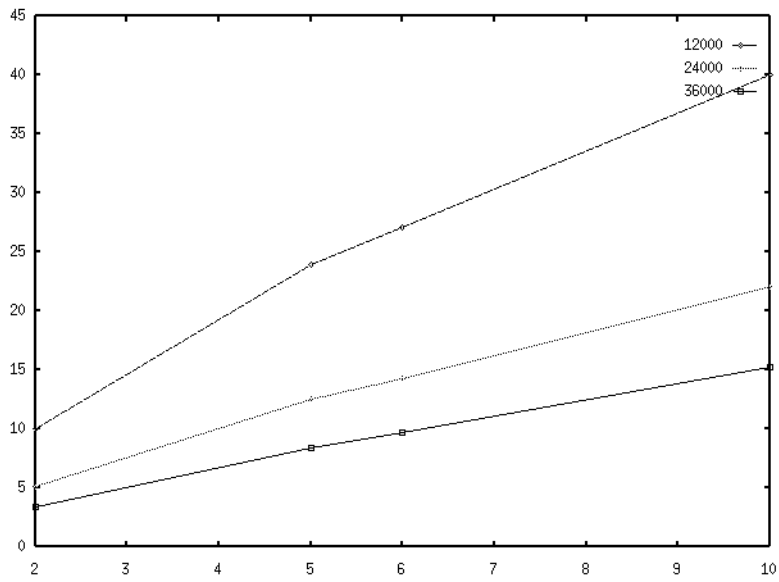


Figure 2: Frame rate as function of Round 0 processors