SmartCam: Devices for Embedded Intelligent Cameras

W. Caarls, P.P. Jonker, and H. Corporaal
{wcaarls, pieter}@ph.tn.tudelft.nl, h.corporaal@tue.nl
Pattern Recognition Group, Department of Electrical Engineering
Delft University of Technology, Eindhoven University of Technology

Abstract—**The advent and subsequent popularity of low cost, low power CMOS vision sensors enables us to integrate processing logic on the camera chip itself, thereby creating so-called *smart sensors*. They have an on-chip SIMD data processing array controlled by an off-chip controller. Smart sensors can execute low-level image processing routines as soon as one or more image lines are converted; they do not have to wait for the whole image. High level image processing like feature extraction and object detection and tracking can be performed with a separate powerful off-chip processor.**

Current solutions have several problems. It is totally unclear what the right architectural parameters are for a given application domain. There are many parameters, like pixel array size, pixel properties, number of AD-converter units, accuracy, number of pixels per SIMD processor element, processor element functionality, etc. Also the functionality of the external processor and its connectivity with the smart sensor has to be determined. Furthermore, intuitive mappings of algorithms on architecture components are used, after the architecture has been determined. It will be clear that this is far from optimal. Finally we foresee a further integration, making a combination of on-chip vision sensor, pixel processing, control, and feature / object processing possible. The result is a low-cost one-chip smart camera (so-called *SmartCam*) solution. This means that research is needed to explore the new architectural opportunities and consequences.

The SmartCam project investigates these new opportunities and contributes to a better and more quantitatively guided design trajectory. In particular, we will investigate the impact of current applications, define relevant architectural parameters and develop an architectural template, enhance our existing application mapping environments for SIMD and ILP (Instruction-Level Parallel) processors, and perform two case studies. The work will focus on creating an environment for exploring the design space parametrized by the architectural template and integrating this with our application mapping environment.

Keywords— **Embedded system design, Smart Sensors, Design Space Exploration, Parametric design, Programmable design, Heterogeneous architecture, Low power, High performance, Small and Cheap implementations**

I. Introduction

Smart CMOS sensors enable low power and integrated intelligence. However, the control processor, feeding the SIMD processor array with instructions and supporting basic program control structures, is usually not integrated with the smart sensor. On top of this, a separate powerful general purpose processor is usually needed in embedded applications for feature and object processing and (motor) control tasks. Integrating all this functionality on a single chip will have a positive effect on the cost, power consumption, latency and inter-processor bandwidth. Given the fact that smart image sensors can already be realized in CMOS and the continuous increase in CMOS density, we foresee that in the future such an integration will take place (see figure 1).

While integrating all this functionality on a single chip can be advantageous, it is very hard to define a solution that fills the needs of a broad range of markets: a designer of AIBO [1]-like appliances will have a very different set of requirements than a designer of high-quality systems for product inspection. Both the SIMD and general purpose processor parts will therefore have to be flexible, and because of this we will focus on using an ILP (instruction-level parallel) general purpose processor, which provides high performance with flexibility tuned to the application at low cost.

Building custom camera-and-logic application-specific integrated circuits (*ASIC*s) (instead of relying on off-the-shelf components such as CMOS vision sensors and general-purpose or digital signal processors) in a sound and informed manner requires a framework in which the design space of these circuits can be explored by the application developer. Guiding the research of the SmartCam project will be the development of an integrated environment for this purpose, and the research topics can therefore be seen as the prerequisites for building such a tool: Application Impact Research (AIR), Architecture Template (AT), Algorithm Transformation Environment (ATE), Mapping and Scheduling (MAS), Analysis Tool (AT), and

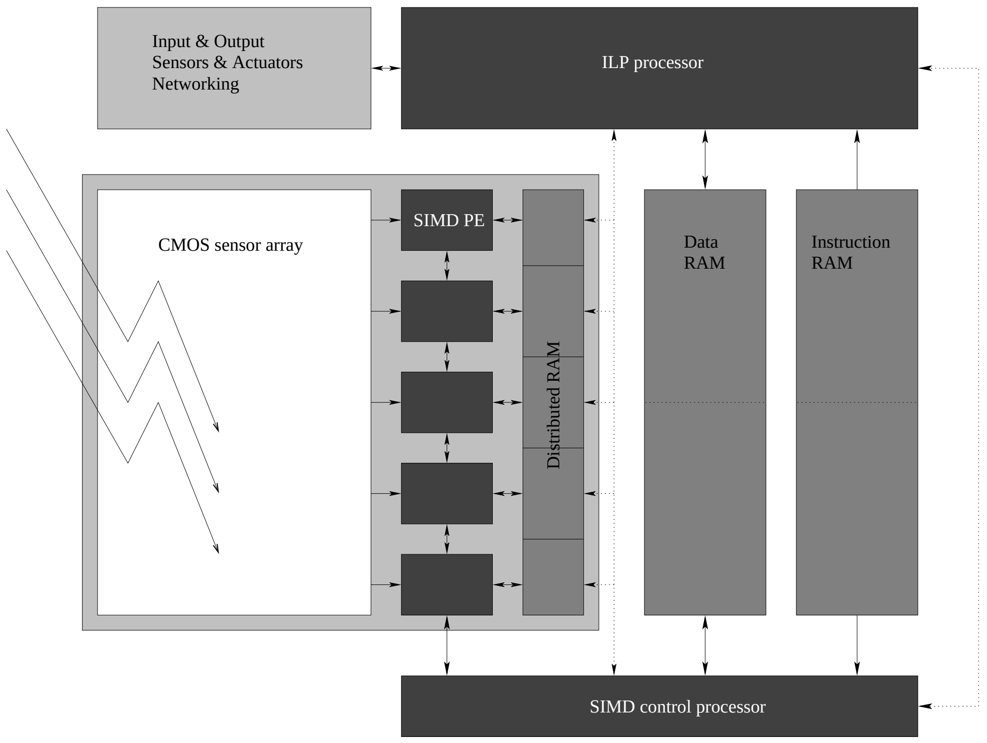

Fig. 1. Basic components of the SmartCam Architecture Template. The dotted lines represent interconnection possibilities which may not be present depending on the architecture parameters.

Design Space Exploration Methods (DSEM). To further evaluate the integrated environment, we will perform two case studies (CS1, CS2).

The rest of the paper will give an overview of these research topics.

II. Application Impact Research

Before building any development tool, it is helpful to have a set of requirements based on the applications that will be built with it. In the case of embedded intelligent cameras, these include parameters such as throughput, latency, memory requirements, image sizes, accuracies, functional requirements, etc. The impact of target applications also includes the inherent ILP parallelism and data parallelism that can be expected, and the determination of a set of core algorithms needed for low- and high level image processing as applied within SmartCams.

For this, we will research the possibilities for SmartCams on several university and commercial applications such as component placement and product inspection. Two university projects will also be used as case studies (see section VIII).

III. Architecture Template

After researching the impact of the target applications, we will define a combined SIMD-ILP architecture template and determine the parameters of interest. The basic structure of this template will depend on the application requirements, but will at least contain a CMOS vision sensor, SIMD processor array, SIMD control processor, ILP processor, distributed and centralized ram, instruction ram, and basic interconnections (see figure 1). Configurable parameters for various parts of the template could then include:

- **CMOS vision sensor** The number of rows and columns of the pixel array, the use of color or grayscale pixel elements, and the number of AD conversion units and their accuracy.
- **SIMD processor array** The number of pixel columns per SIMD processing element (PE), the number of bits of each PE, PE functionality, distributed memory size and organization, SIMD control processor (CP) functionality.
- **ILP processor** The number of function units (FUs), operation repertoire (determining the type of function units), the number of registers, register files and ports, FU connectivity, memory hierarchy and cache sizes.
- **Interconnections** between the vision sensor and PEs, PEs and distributed RAM, PEs amongst themselves, CP and RAM, ILP and CP, ILP and RAM. Note that it is possible to have quite complex sensor-PE interconnectivity for applications that use specific regions-

of-interest (*ROI*s) and that want to address those in a linear manner, requiring a translation to a *virtual* sensor array.

The architecture template will also have to incorporate the possibility of *not* integrating the CMOS vision sensor with the processing logic. This is necessary to accommodate high-resolution sensors with powerful computation, which is not possible on a single chip due to area and process restrictions. Since the CMOS process parameters of the sensor and logic are not the same, some tradeoff between these two will have to be made, and if no satisfactory solution is possible a multi-chip solution must be considered.

IV. Algorithm Transformation Environment

In order to make efficient use of the possibilities of the SIMD and ILP hardware, the structure of the application (assumed to be written in C/C++) may need to be transformed. Programmers are usually more concerned with functional correctness than with the possible mapping of their software onto specific hardware components, and especially because of the fluidity of the hardware due to our architectural template they must be assisted in this mapping.

First, we will assist the application developer by providing a library of implementations of kernel image processing algorithms (like the one developed in the NWO-PILE program [6]) which are parametrized in much the same way as the architectural template. If the developer uses these algorithms in his software, it will automatically use the best hardware mapping available within the library.

Second, we will use and extend a source code transformation tool (CTT, [3]) to assist the developer in enhancing loop parallelism, increasing the scheduling scope, removing harmful dependencies, better using the memory hierarchy or exploiting local memories, and reducing power requirements.

V. Mapping and Scheduling

After defining the architecture template, we can use a specific parameter setting on that template and the transformed code to generate a mapping of the application on the hardware, and to schedule the use of the available hardware (function units, registers, buses, etc.). This requires the use of a retargetable compiler, and we will make use of our previous work in the field of ILP and SIMD code generation ([3], [4]).

Since this step will be the basis of our performance evaluation, the quality of the generated code will be crucial. Our goal is to create a compiler environment that can systematically cope with all the different hardware parameters. During the exploration phase it is possible to further augment the performance of the compiler by using the execution profile generated in the analysis stage, described below.

VI. Analysis Tool

The analysis tool supplies both a performance and cost evaluation. The performance estimation depends on the clock speed of the generated hardware and the number of machine cycles it takes the application to execute a predetermined input or set of inputs. In both cases it is infeasible to generate a complete layout or to do a full machine simulation, so we will have to use analytic models for predicting relevant parameters like the length of the critical path. For estimating the execution time, it can be helpful to first generate a high-level execution profile in terms of loop repetitions, branch probabilities, etc. This speeds up the analysis time significantly, and can still give cycle-accurate results [5].

As with the cycle time estimation, it is infeasible to generate a full layout for the evaluation of the needed chip area and power dissipation. We will research analytic models specifically for the developed SmartCam architecture template environment. Such a model will also be necessary to predict the memory behavior of the ILP because of cache indeterminacy.

VII. Design Space Exploration Methods

The final prerequisite for building a SmartCam design space exploration tool is the development of algorithms to steer the possible software, hardware and tool chain transformations (see figure 2). Even when restricting the design space to our template, and to a well defined set of source-to-source transformations, the design space is extremely huge and cannot be investigated manually or exhaustively. Many algorithms and AI techniques have been used for this purpose ([5], [2]), and we will be using these to steer the transformations semi-automatically: we believe the designer always has final control, and can use his application knowledge to drastically reduce the design space.

VIII. Case studies

We will use two case studies to evaluate and further refine our design environment: the UBICOM-Headset and RoboCup projects.

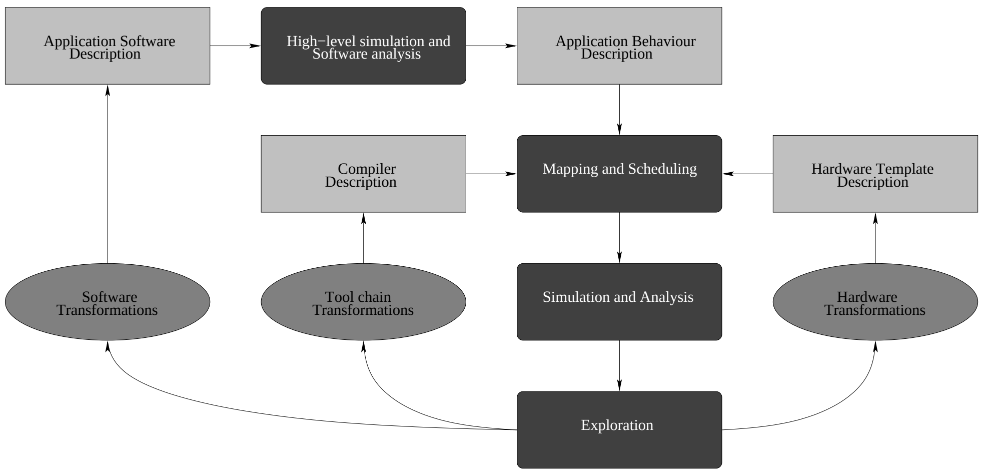
Application Software Description
High−level simulation and Software analysis
Application Behaviour Description
Compiler Description
Mapping and Scheduling
Hardware Template Description
Software Transformations
Tool chain Transformations
Simulation and Analysis
Hardware Transformations
Exploration

Fig. 2. Design Space Exploration flow diagram, based on the Y-chart philosophy.

A. *UBICOM-Headset*

Within a current program of the TU Delft, The Ubiquitous Communication Project (UBICOM), augmented reality headsets based on retinal scanning display techniques are involved [7]. With such a headset on, persons can have a virtual image projected exactly over their normal view to augment their reality. The graphics involved comes over a wireless link from some backbone system. However, the graphics should be adapted to the current position of the person's head and in case of the retinal scanning display technique, also to the position of the eye pupil.

For such a system, low cost tracking camera's are needed that are able to track objects to maintain position and orientation of the user and very high speed but small array camera's are needed to keep track of the pupil positions.

B. *RoboCup*

The RoboCup project aims to stimulate research into autonomous intelligent agents by organizing soccer competitions between robots. Challenges include localization, object recognition, control tasks, artificial intelligence strategies, team cooperation, etc.

The use of a smart camera for segmentation, object detection and tracking at high speed allows for a faster cycle time and better control. It also releases the central processor of these low-level tasks such that it can devote more time to high-level AI decisions.

IX. Conclusion

We have introduced the SmartCam project and discussed the necessary steps to enable application developers to create custom smart sensors in a quantified and informed manner. We will realize this by specifying an architecture template for camera-and-logic applications and building an integrated development environment for exploring the design space of that template. The development environment will feature analysis tools that provide feedback to the developer about the chip area, power consumption, and speed of a proposed solution, allowing him to make the final cost/performance tradeoff.

This work is supported by the Dutch government in their PROGRESS research program.

References

[1] Entertainment robot aibo. Website. http://www.aibo.com.
[2] G. Ascia, V. Catania, and M. Palesi. A framework for design space exploration of parametrized vlsi systems. In *Proceedings of the 15th International Conference on VLSI Design*. IEEE Computer Society, 2002.
[3] M. Boekhold, I. Karkowski, and H. Corporaal. A programmable code transformation engine. In *Proceedings of ETAPS99*, 1999.
[4] H. Corporaal. *Microprocessor architectures: From VLIW to TTA*. John Wiley and Son Ltd, 1998. ISBN 0-471-97157-X.
[5] G.J. Hekstra, G.D. La Hei, P. Bingley, and F.W. Sijstermans. Trimedia cpu64 design space exploration. In *Proceedings of the IEEE International Conference on Computer Design*, pages 599–606. IEEE Computer Society, 1999.
[6] C. Nicolescu and P.P. Jonker. Towards parallel image processing in heterogeneous architectures. In H. Arabnia, editor, *Proceedings of the International Conference on Imaging Science, Systems, and Technology*, pages 311–317. CSREA Press, 1999.
[7] S. Persa and P.P. Jonker. On positioning for augmented reality systems. In H.-W. Gellersen, editor, *Proceedings of the First International Symposium on Handheld and Ubiquitous Computing*, volume 1707 of *Lecture Notes in Computer Science*, pages 327–329. Springer, Berlin, 1999.