Automatic Mapping of Khoros-based Applications to Adaptive Computing Systems

S. Natarajan, B. Levine, C. Tan, D. Newport and D. Bouldin
Electrical and Computer Engineering, University of Tennessee
Knoxville, TN 37996-2100
dbouldin@utk.edu

Abstract

An adaptive computing system (ACS) composed of programmable logic components can serve as a flexible hardware accelerator, especially for image-processing applications. To enable application designers who are unfamiliar with hardware details to map their applications onto an ACS, a software environment called CHAMPION is being developed. This paper presents the specific steps required to map a Khoros flow diagram onto an ACS constructed using five Xilinx XC4000 parts. A productivity improvement of 100x (2.5 hours vs. 250 hours) was estimated for the automatic implementation as compared to a manual one for a moderately complex target recognition algorithm. The ACS was able to execute the algorithm 156 times faster than the Khoros simulation running on an UltraSparc.

I. Introduction

Traditionally the mapping of an application onto an ACS has taken months to develop and debug. The goal of the research described here is to develop and demonstrate a software design environment which will perform this mapping automatically, resulting in a 100x improvement in productivity. The environment permits high-level design entry using the popular Khoros Cantata visual programming software[1-2]. Thus, application designers need not be familiar with the details of the hardware. Khoros was developed over ten years ago, is available free over the web, and is in use by over 5000 designers world-wide. It enables a designer to capture an algorithm quickly, make changes in which operations are performed and in which order and then see the results in a few minutes without having to develop C code.

To implement an algorithm using Khoros, an image-processing designer develops a workspace by connecting together Khoros glyphs (library cells) as shown in Figure 1. These glyphs can be thought of as C subroutines, while the Khoros workspace can be viewed as a main C program that invokes these subroutines and controls the data flow from one subroutine to another.

Therefore, the objectives of the CHAMPION translation software are to parse a Khoros workspace and automatically translate it into a form that can be executed on an ACS containing multiple programmable logic components.

II. Library Cell Development and Verification

A Khoros designer who wishes to use CHAMPION must first install the CHAMPION cell library and then develop his workspace using only these cells (glyphs). The CHAMPION library contains 30 cells that are commonly employed by image-processing applications. Additional cells will be added as needed when CHAMPION is used for other applications of interest.

These CHAMPION cells differ from those normally provided with Khoros since each cell performs only one specific function on one data type using a fixed (but parameterized) bit width. Furthermore, the CHAMPION cells provide low-level mechanisms for synchronization with other cells that are not traditionally required in a Khoros workspace.

Each of the CHAMPION cells has been derived from a VHDL source code file that has been synthesized (using commercially available tools from Synopsys or Synplicity) into the target devices (Xilinx XC4000 series in this case but the Altera FLEX 10K and Xilinx Virtex versions will be completed this Fall). Using CHAMPION-specific cells that have been pre-compiled enables CHAMPION to determine quickly the size requirements of the application and the number of clock cycles required to perform individual operations. This information is then used to match data bit-widths, synchronize delay paths and efficiently partition the resulting flow graph into subcircuits. These subcircuits are then placed and routed using commercially available tools (Xilinx M1 and Altera MaxplusII) and downloaded into the ACS.

It is therefore necessary to ensure the same functionality for the CHAMPION glyphs in Khoros, the VHDL source files and their corresponding programmable logic configuration files. The first step towards achieving this goal is defining precisely a glyph's parameters and its explicit functionality. A Khoros glyph is then developed using a high level programming language such as C++. Input test vectors are then applied to this glyph and the output responses are stored. These input and output vectors together constitute a test bench that can be used for computer simulations of the glyph. The synthesized circuit is simulated with the previously generated test bench to ensure the validity of the technology-specific library. At this stage, the programmable logic configuration file is ready for execution on the ACS platform. Verification of the hardware execution entails applying the same test bench to the hardware and observing its outputs. The hardware glyph passes this final verification stage when its results

concur with those of the computer-aided simulations. Once verified, the hardware glyph is characterized in terms of size and number of clock cycles (and eventually power consumption).

III. Khoros Image-Processing Applications

The sponsor of our research project, DARPA, has asked us to develop CHAMPION and test it using three significant challenge problems and on three different ACS platforms. Since it is well known that ACS platforms offer the best price/performance solution for numerous image-processing applications [3], we are collaborating with the Army Night Vision Lab (NVL) on an automatic target recognition application as our initial challenge problem. The Khoros workspace consists of almost 1000 glyphs but only about 25 unique library cells.

To prepare CHAMPION for the NVL challenge problem, a simpler yet moderately complex algorithm was selected. This algorithm was also translated manually so that the steps to be automated by CHAMPION could be delineated. This approach also provided a baseline to quantify the productivity improvement provided by the new software.

The initial application chosen was an automatic target recognition application called START that was developed in-house and the initial ACS platform used was a commercial product called the Wildforce-XL from Annapolis Micro Systems [4]. Using a complex application provided several benefits. It helped identify more useful hardware glyphs than a simpler algorithm. Using an algorithm large enough to require reconfiguration of the board also tested that capability of the board and required the determination of techniques to partition the design temporally as well as spatially. Finally, using a complex algorithm validated that the libraries and procedures developed were sufficient to complete problems of a significant nature.

The START application uses a statistical algorithm to find regions in Forward-Looking InfraRed (FLIR) images where a target may exist and draws a box around such regions. Algorithms of this sort are often used for target queuing; that is, they automatically identify areas of possible interest for further examination by human operators. The algorithm does not positively identify a target, nor does it identify the type of target, as a true ATR algorithm does. It simply identifies segments of the image as areas where there is a high probability of there being one or more targets.

The START algorithm was tested with FLIR images from Colorado State University's Fort Carson RSTA Data Collection. This is a freely accessible collection of image data available over the Internet [5]. Forty images were chosen from the entire set of available images. The images chosen had corresponding color visual light images available so that the actual location of targets could be determined more accurately, as identifying targets in the FLIR image can sometimes be difficult for a human observer. The FLIR images were taken of varied, generally hilly terrain, with either no vehicles present, or up to four vehicles present. The vehicles used to represent targets were an M60 main battle tank, an M113 armored personnel carrier, an M901 anti-tank vehicle, and a GMC pickup truck.

Areas of interest are assumed to have two characteristics. First, they must contain pixels hotter than the surrounding terrain. In a FLIR image, hotter pixels are brighter or higher in numeric value. An area of interest must also contain pixels having a high numerical value after the application of an edge detection filter. A high numerical value indicates a large gradient intensity between adjacent pixels, which corresponds to a sharp temperature gradient between adjacent regions. Vehicles are likely to exhibit sharp temperature gradients, either between the vehicle and the surrounding terrain, or between different components of the vehicle, such as an exhaust port and the chassis. An area of interest must satisfy these two different criteria: it must contain hot pixels and it must contain pixels that exhibit large gradients. These are the two criteria in the name START (Simple, Two-criterion, Automatic Recognition of Targets).

While the core idea of the algorithm is relatively simple, many details of the START algorithm are important to its success. First, the images are low-passed filtered initially to remove noise, as image noise is generally high frequency in nature and produces strong responses in edge detection filters. The threshold values for each of the two criteria are specified as a certain number of standard deviations from the mean. These values are important to the performance of the algorithm and were determined empirically during processing of a large set of input images. The intermediate binary images in the target identification process are low-pass filtered and then thresholded. This favors larger clusters of pixels and eliminates the occasional falsely identified pixel.

The steps involved in mapping a Khoros application onto an ACS will be described next. The flow is shown graphically in Figure 2.

IV. Data Width Matching and Synchronization

A direct translation from Khoros to an ACS platform is not possible. There is much more involved than converting one data structure or format into another. Obviously, the initial step is straightforward in that it consists merely of changing the Khoros workspace or network into a more graph-oriented one. But when the graph is analyzed, we noticed numerous arithmetic operations which produce results that require more bits for their outputs than for their inputs. Consequently, glyphs originally cascaded to one another will progressively require a wider data path to avoid roundoff errors. When one path of operations is then connected to a parallel path, a mismatch in the number of bits for these inputs may occur. Therefore, one required step in mapping the Khoros workspace is to analyze each operation and insert the capability for additional bits when appropriate.

Another observation gleaned from the manual implementation of the START workspace was the realization

that data traveling over different concurrent paths may arrive at the input of an operator at different times. In order that the data inputs be synchronized, the number of clock cycles required for each path must be enumerated. Then a compensating delay can be inserted in the shorter path to balance them in time. The graph shown in Figure 3 depicts a section of the application and shows the initial nodes along with those that were added for data matching and synchronization.

The two steps just described are very straightforward but are also very time-consuming and tedious for a human to perform. In the case of START, almost half of the time required to map it onto the ACS was consumed just by these two steps. Having a computer program perform these steps relieves a human from these boring and error-prone duties and can improve the productivity of the conversion process from 100 hours or so down to only a few minutes.

V. Partitioning

Our approach to the partitioning problem is based on the variant of standard move-based bipartition heuristics [6-8]. The multi-way partitioning is achieved by recursively applying the bipartitioning algorithms to the netlist of the design until it is split into the required number of sub-netlists. The approach to this partitioning problem is similar to the method described in [8].

For our first implementation, we configured the programmable logic components and their interconnect on the Wildforce board into a linear array. With this board topology, the multi-way partitioning order proceeded in a forward direction. The first bipartition split the netlist of the Khoros workspace into two unbalanced sub-netlists such that one of the sub-netlists met the constraints on size and the number of pins of the first programmable logic part. Then we again applied the same bipartition technique to the remaining sub-netlist to obtain the second partition, which was then mapped to the second programmable logic part. We continued applying this bipartition technique to obtain sub-netlists for the remaining parts.

The results obtained using this unbalanced bipartition technique produced the same results as those determined manually. With this straightforward method, partitioning was accomplished in only a few minutes automatically whereas partitioning performed manually required several hours of effort. More complicated combinations of ACS architecture configurations and application graphs will likely be extremely difficult for both man and machine. Hence, we are planning to provide a suite of partitioning algorithms which could be selected by the application designer.

VI. Final Conversion Steps

After partitioning, the internal data structure or format is then translated into structural VHDL. The resulting file can then be passed quickly through a synthesis tool to add the required I/O ports and to merge pre-compiled VHDL components corresponding to the Khoros glyphs. Just as some of the previous steps were found to be straightforward but time-consuming and tedious for a human to perform, the final conversion of the partitioned graphs into structural VHDL files was laborious. Fortunately, no heuristics or complex algorithms are needed and the time savings is significant since tens of hours can be reduced to only a few minutes.

Once the structural VHDL files have been generated, each one is synthesized separately, then placed and routed and the configuration file downloaded to the corresponding programmable logic component on the ACS Wildforce board.

VII. Results and Discussion

Executing the partitioned START algorithm on our Wildforce ACS resulted in a speedup of 156 as compared to the Khoros simulation executing on an UltraSparc workstation. The individual tasks comprising the execution included: (1) software execution on the CPU host, (2) configuration of the programmable logic components, (3) data transfer time (from the host RAM to and from the local RAM), and (4) the actual data processing time on the ACS. Timing varied according to server load, but the average of 50 runs is shown.

It is immediately obvious that for our Wildforce board running an application of this complexity, the time to process one image is greatly dominated by the time needed to configure the board. The actual time to process one image is only 33 milliseconds, as compared to the nearly seven seconds needed for the entire execution. However, the hardware implementation was still more than 156 times faster than the Khoros simulation on the CPU host. If sufficient logic and memory resources were available such that the reconfiguration time could be eliminated, the hardware implementation would be 667 times faster than the Khoros simulation. Finally, if the hardware was configured to process images sequentially, with no setup time or configuration time necessary, then the ACS hardware implementation would be over 32,000 times faster than Khoros simulation.

VIII. Conclusions

A moderately complex image-processing application was captured using Khoros and a set of library cells that had been pre-compiled for programmable logic components. The resulting workspace was simulated on an UltraSparc and then converted manually into a graphical data structure. This graph was then altered to match data bit-widths and to synchronize delay paths. The graph was then partitioned using an unbalanced bipartitioning technique for a linear array of programmable logic components. The partitioned sub-graphs were then translated into structural VHDL files. Each file was synthesized and then placed and routed to produce configuration files for the programmable logic components. Since there were insufficient resources on the ACS platform to handle this application, the programmable components were

reconfigured as needed to produce the final result. This reconfiguration time dominated the ACS execution and pointed to the need for more logic and memory resources and/or faster reconfiguration times for programmable components. The productivity improvement estimated for CHAMPION is expected to be at least 100x since the manual implementation took 250 hours but the automatic conversion should take less than 2.5 hours.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the support of DARPA grant F33615-97-C-1124.

REFERENCES

[1] J. R. Rasure and C. S. Williams, “An Integrated Data Flow Visual Language and Software Development Environment”, *Visual Languages and Computing*, vol. 2, pp. 217-246, 1991.

[2] J. R. Rasure and S. Kubica, “The Khoros Application Development Environment”, Khoros Research Inc., Albuquerque, NM, http://www.khoral.com.

[3] J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, and P. Boucard, “Programmable Active Memories: Reconfigurable Systems Come of Age,” *IEEE Trans. On VLSI Systems*, vol. 4, no. 1, pp. 56-69, March 1996.

[4] Annapolis Micro Systems, Annapolis, MD, http://www.annapmicro.com.

[5] Colorado State University Computer Vision Group, Fort Carson RSTA Data Collection, http://www.cs.colostate.edu/~vision/ft_carson.

[6] C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions", *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pp. 175-181, 1982.

[7] S. Hauck and G. Borriello. "Logic Partition Orderings for Multi-FPGA Systems", *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, pp. 32-38, February, 1995.

[8] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning of Electrical Circuits", *Bell Systems Technical Journal*, Vol. 49, No. 2, pp. 291-307, February 1970.

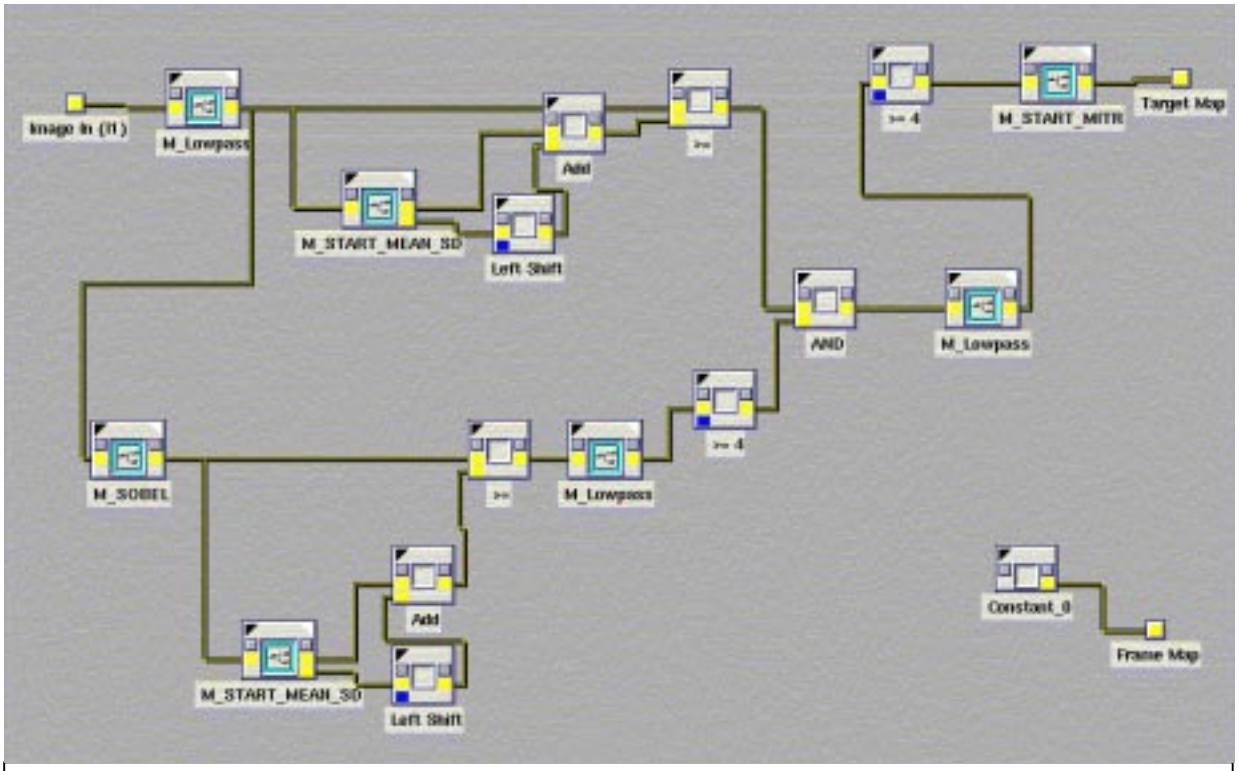

Figure 1: Khoros visual programming application workspace.

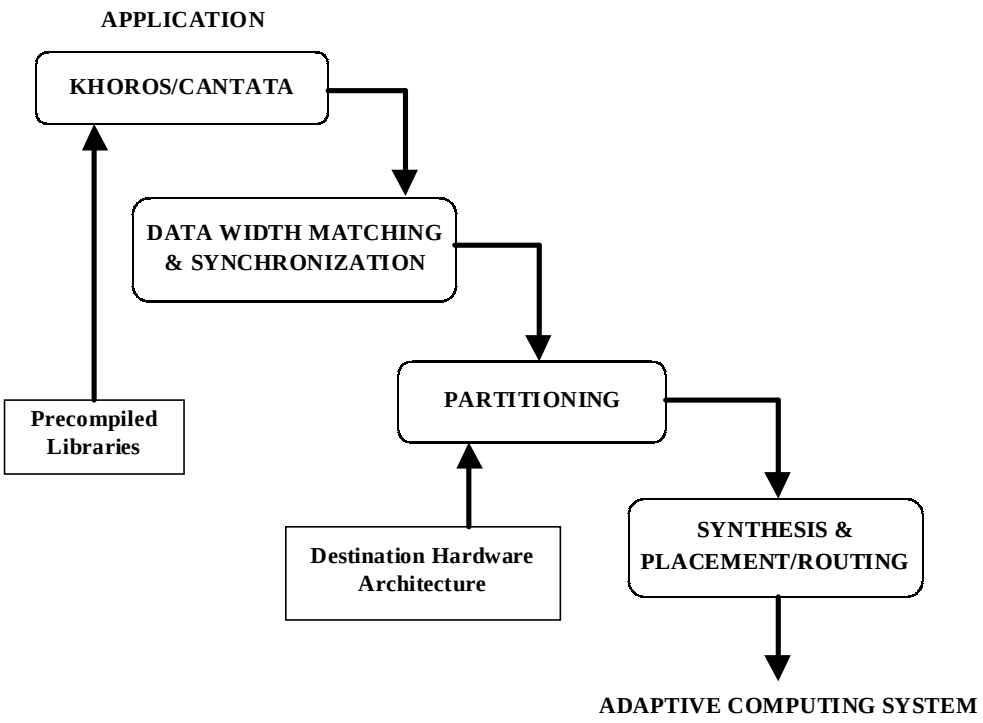

Figure 2: Steps required to map Khoros workspace onto an adaptive computing system.

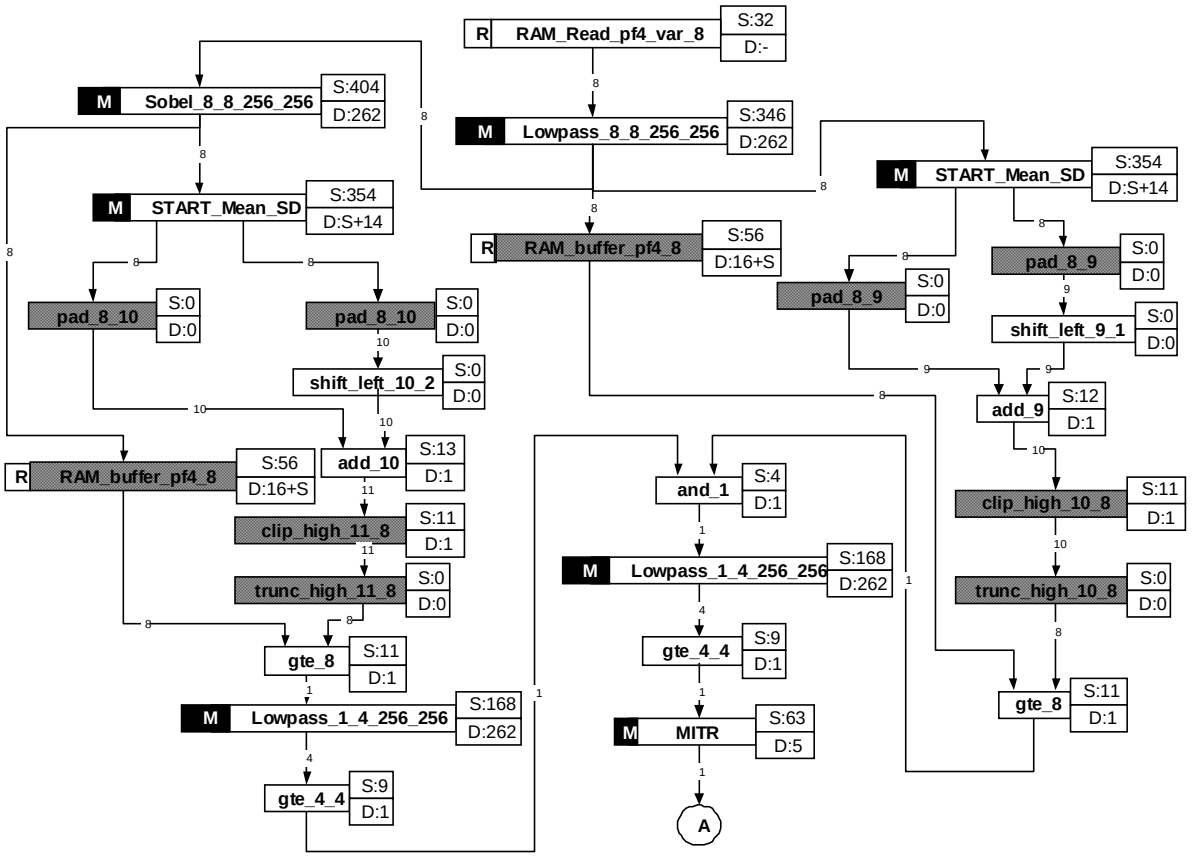

Figure 3: Hardware task graph with data bit-width matching and synchronization.