Design Flow for Automatic Mapping of Graphical Programming Applications to Adaptive Computing Systems

S. Ong, N. Kerkiz, B. Srijanto, C. Tan, M. Langston, D. Newport and D. Bouldin
Electrical and Computer Engineering, University of Tennessee
ong@utk.edu

Abstract

Graphical programming environments such as Khoros [1] from KRI, LabVIEW from National Instruments and Simulink from MathWorks, separate application-level programming from low-level programming by allowing programs to be constructed by interconnecting precompiled programs or functions. These programming environments allow faster and easier development of complex applications; however, the execution times for image and digital signal processing are often long due to large input data sets and computationally intensive processing functions. With the advent of adaptive computing systems (ACS), these applications which are traditionally implemented in software can now be implemented in reconfigurable processing devices. As a result, the ACS can serve as a flexible hardware accelerator for time-consuming and computationally intensive applications. However, programming an ACS remains a significant obstacle to its widespread adoption by application programmers who are generally not familiar with hardware design. To enable designers to map their applications onto an ACS, a software design environment called CHAMPION is being developed. This paper presents the algorithms used in CHAMPION for mapping Khoros programs onto ACSs which utilize multiple FPGA-based architectures.

LIBRARY CELL DEVELOPMENT AND VERIFICATION

An overview of the design flow of CHAMPION is shown in Figure 1. Khoros is used as a function oriented programming environment where all the application programs are developed using predefined functions called glyphs. These primitive functions are part of the CHAMPION precompiled library developed using fixed-point C or C++. Each of these glyphs is derived from a VHDL source code file that has been synthesized using commercially available tools into the target ACS architecture.

Additional cells can be added to the precompiled library as needed when CHAMPION is used for other applications of interest. To incorporate a new cell into CHAMPION, the designer must first develop the C or C++ program and the corresponding VHDL code for the cell. For complex functions, the C or C++ programs and VHDL codes can be formed as macros of lower-level functions. The Khoros tools can then be used to transform the C or C++ program into a Khoros glyph. For the installation of the VHDL hardware cell into the CHAMPION library, a CHAMPION software tool can be used. The CHAMPION tool automates the synthesis of the VHDL code using the commercial logic synthesis tool, *Synplify*. Based on the ACS architecture specified by the designer, the tool will generate the required technology-dependent file and a text file (named *inf* file) for storing the information such as the size, latency and I/O data bit-widths of the hardware cell. This information will be used during the data width matching, data synchronization and partitioning processes.

CONVERTING KHOROS WORKSPACE TO CHAMPION NETLIST

Using Khoros, the designer can develop the application by interconnecting CHAMPION glyphs to form the Khoros workspace. Simulation, data analysis and visualization can then be performed in Khoros on the UNIX workstation. Once the desired functionality of the application is achieved, the Khoros workspace can be automatically mapped onto the target ACS. The initial step in CHAMPION consists of translating the Khoros workspace into a more graph-oriented netlist format. The netlist format is a directed hypergraph where each glyph is represented as a node and the interconnections between glyphs are represented as directed hyperarcs. Based on the information from the *inf* file, weights are assigned to the nodes and hyperarcs of the directed graph. The weights of the nodes correspond to the size in terms of the number of logic blocks in the FPGA, and the weights of the hyperarcs correspond to the net-width of the glyph interconnections. This netlist format simplifies the use of graph theory and network optimization theory during the data synchronization process presented in the next section.

DATA WIDTH MATCHING AND SYNCHRONIZATION

In a Khoros application, some Khoros functions may 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 connected to a parallel path, a mismatch in the number of bits for these inputs may occur. A software tool in CHAMPION analyzes each operation and inserts additional bits when appropriate.

In the graphical programming environment such as Khoros, executions of the programs are data driven. That is, each glyph will execute only when all its input data is available. However, hardware systems are clock driven. At each clock cycle each hardware cell will process whatever data is presented at its inputs.

Due to the difference in the processing time of each hardware cell, data traveling over different concurrent paths may arrive at the inputs of a multi-input cell at different times. To insure that each cell generates the correct time-sequenced output, it is necessary that each cell receive all its input data precisely at the same time. This requirement is often referred to as data synchronization. In CHAMPION, data synchronization is achieved by introducing delay buffers into the system. The synchronization software determines the lengths and locations of the delay buffers necessary to balance the various data paths. The optimization algorithm in [2] is employed to calculate a set of buffer lengths and insertion points that maximizes the amount of buffer sharing and therefore minimizes total system delay. Minimizing the sum of the lengths of such delay buffers is desirable in order to reduce the hardware utilization.

PARTITIONING

In CHAMPION, the partitioning problem is based on the variant of standard move-based bipartition heuristics [3-5]. The multi-way partitioning is achieved by recursively applying bipartitioning 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 [5]. In the Annapolis Micro Systems Wildforce ACS, the programmable logic components and their interconnects are configured into a linear array. With this topology, the multi-way partitioning order proceeds in a forward direction. The first bipartition splits the netlist into two unbalanced sub-netlists such that one of the sub-netlists meets the constraints on size and the number of pins of the first programmable logic component. The same bipartition technique is then applied to the remaining sub-netlist to obtain the second partition, which is then mapped to the second programmable logic component. This bipartition technique is continued to obtain sub-netlists for the remaining components. Other partitioning algorithms are currently being implemented. Eventually, a suite of partitioning algorithms will be available in CHAMPION for the application designer to select.

NETLIST TO STRUCTURAL VHDL, SYNTHESIS, PLACE/ROUTE AND HOST PROGRAM

After partitioning, the graph-based netlist format is translated into structural VHDL. The resulting files are passed through a synthesis tool to add the required I/O ports and to merge the pre-compiled VHDL components corresponding to the Khoros glyphs. Once the structural VHDL files have been generated, each structural VHDL file for each of the programmable logic component is synthesized, and then placed and routed separately. A CHAMPION software tool will then generate the host program to download the configuration file to the corresponding programmable logic component on the ACS.

CONCLUSION

With the advent of ACS, algorithms that have been traditionally performed in software can now be implemented in reconfigurable processing devices. However, programming these ACSs remains one of the major obstacles to the widespread adoption of the ACS. The main driving force of the CHAMPION project is to shift the process of mapping the application onto an ACS from being hardware-centered to being software-centered.

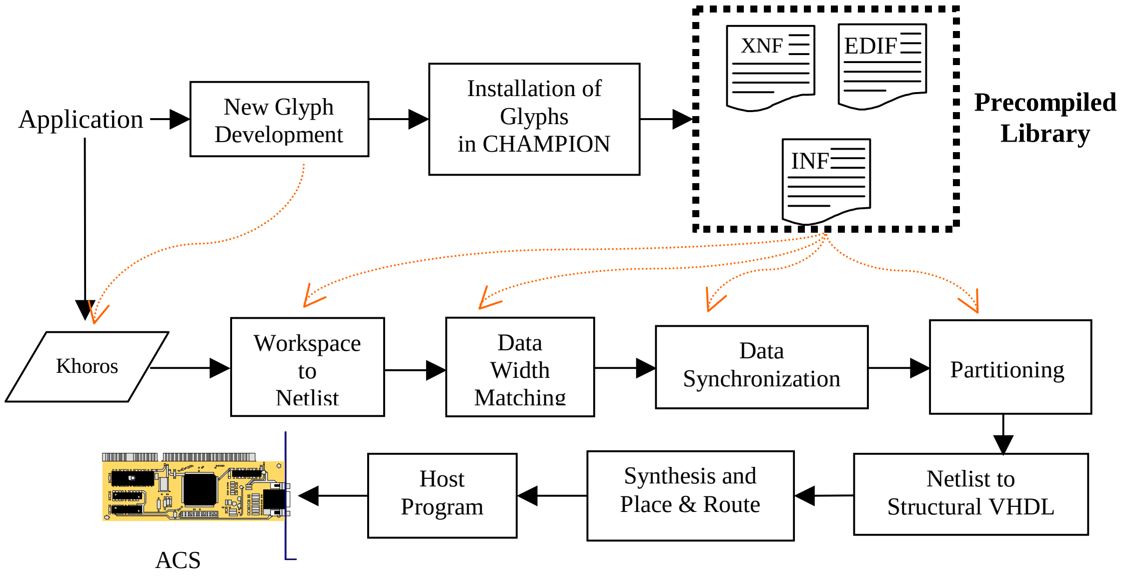

Figure 1. Overview of the design flow in CHAMPION.

REFERENCES

[1] J. Rasure and S. Kubica, *The KHOROS Application Development Environment*, Khoros Research Inc., Albuquerque, NM, http://www.khoral.com.

[2] X. Hu, S. C. Bass and R. G. Harber, “Minimizing the number of delay buffers in the synchronization of pipelined systems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, No. 12, pp.1441-1449, Dec 1994.

[3] 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.

[4] 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.

[5] 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.

ACKNOWLEDGEMENT

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