

Run-time Mapping of Graph-Problem Instances onto Reconfigurable Hardware*

(Summary)

Andreas Dandalis, Viktor K. Prasanna, and Jean-Luc Gaudiot

University of Southern California
{dandalis, prasanna, gaudiot}@ceng.usc.edu
Tel: +1-213-740-4483, Fax: +1-213-740-4418

The Problem

During the past few years, the rapid advances in fabrication technology has led to the development of programmable devices (e.g., FPGAs) with substantial computational power. As a result, reconfigurable hardware is being used beyond the initial applications of rapid prototyping and emulation into several areas of general purpose computation. Existing evidence suggests that using programmable devices for DoD applications will result in performance capabilities that are 2-3 orders of magnitude better than technologies currently being used [2].

Currently used military platforms are mainly based on ASICs to meet the real-time constraints and computational demands in battlefield environments. Reconfigurable Computing (i.e., computing using programmable hardware) can “outperform” ASICs by exploiting its ability to create hardware at runtime based on input parameters. If the logic remains static for all the instances of the problem, then an ASIC implementation would provide superior time performance. In addition, the essential struggle against time in the battlefield necessitates that the hardware adaptation has to be performed very fast. However, existing mapping techniques require extensive mapping time which is a major bottleneck in the case of any mapping that needs to be performed at runtime based upon the problem instance.

Existing mapping techniques for FPGAs have adopted the ASIC-based design flow and tools that prevent the Reconfigurable Computing paradigm from achieving its full potential: provide the performance benefits of ASICs and the flexibility of microprocessors. For one thing, current design compilation times are too long and preclude any run-time, dynamic modification of the configurations. For another, the characteristics of the application are not utilized, resulting in sub-optimal designs with respect to area

and delay performance unless the designs are optimized by hand.

Our Approach to Run-time Mapping

Most of the mapping techniques proposed in the literature ignore the extensive overhead of the CAD tools at run-time. We believe however, that addressing this overhead is the key to fully exploit the Reconfigurable Computing advantages over ASICs and software based approaches.

Our approach to run-time mapping is to handle the mapping problem as an algorithm synthesis problem as opposed to “stuffing logic into a black box.” Our key idea is to develop problem-specific configurations off-line to facilitate run-time mapping. These configurations are specific to the problem to be solved and are based on the algorithm that is used to solve the problem. At runtime, a mapping algorithm adapts the hardware to the input problem-instance. Our performance metric includes the time to compute the logic to be mapped, the time to configure the hardware, and the execution time on hardware.

The novelty of our approach is that the CAD tools bottleneck is alleviated from the critical path to the solution. The mapping process is driven by problem-specific configurations that are derived off-line. Thereby, there is no need for a complete redesign for each problem-instance. Equally important, the mapping process is aware of the characteristics of both the problem and the target architecture. Not only does the approach significantly speed up run-time mapping but also produces fast, compact logic reducing execution time as well. Preliminary results indicate that our approach can result in a speedup of at least two orders of magnitude comparing with the state-of-the-art.

A Case Study: Single-Source Shortest Path Problem

In our current efforts, we are focusing on mapping graph-problem instances onto multi-FPGA systems. Graph problems are the most frequently solved class of

*This research was performed as part of the MAARC project (<http://maarc.usc.edu>). This work is supported by the DARPA Adaptive Computing Systems program under contract no. DABT63-96-C-0049 monitored by Fort Haachuca.

Problem size # vertices x # edges	Clock rate (MHz)		Execution time (μ sec)		Mapping time		Effective Speed-up
	[1]	Our solution	[1]	Our solution	[1] +	Our ++ solution	
16 x 64	1.79	15	8.94	21.42	~ 4 hours	~ 22 msec	6.5×10^6
64 x 256	1.14	15	56.14	79.02	~ 4 hours	~ 82 msec	1.7×10^6
128 x 515	0.78	15	164.10	199.72	~ 8 hours	~ 161 msec	1.8×10^6
256 x 1140	0.34	15	752.94	493.17	~16 hours	~ 319 msec	1.8×10^6

+ a cluster of 10 workstations was used
++ memory-array bandwidth 4MB/sec is assumed as in [1]

Table 1: Performance comparison with the state-of-the-art

optimization problems (e.g., problems of heuristic search, deterministic optimal control problems, or data routing within a computer communication network).

In the state-of-the-art technique for solving graph problems using FPGAs [1], the input graph instance is embedded in the FPGAs by using general purpose CAD tools. A complete redesign is required for a new problem instance and the resulting implementation lacks modularity. Partitioning and place-and-route take several hours while the corresponding execution time on hardware is in the range of μ sec. However, the mapping time is usually ignored and only the execution time is considered as runtime.

Besides the mapping overhead, the mapping of edges onto the physical wires of a device results in extremely slow clock rate and very high area requirements. The clock rate depends on the longest wire in the layout. As the number of vertices increases, the longest wire length increases rapidly resulting in fast degradation of the clock rate. Also, the area requirements depend on the connectivity of the input graph and increase rapidly for dense graph instances. Therefore, the clock rate and the area requirements cannot be reliably estimated before actually mapping onto hardware.

To illustrate the superiority of our ideas, we briefly describe a solution for the single-source shortest path problem using our approach and compare it against the state-of-the-art (based on CAD tools). Given a weighted, directed graph and a source vertex, the problem is to find a shortest path from the source to every other vertex.

A problem-specific configuration is developed based on the Bellman-Ford algorithm. The configuration corresponds to a general graph with n vertices and e edges and consists of n modules connected in a pipelined fashion. Each module corresponds to a vertex. At runtime, the problem-specific configuration is adapted to the characteristics of the input graph instance. At the module level, the precision of the functional units is adapted to the precision requirements and the number of vertices and edges of the input graph instance. Moreover, at the layout level, the number of the modules mapped onto hardware is determined by the number of vertices in the input instance. Finally, the clock speed is determined by the computational

rate of the modules and the available I/O bandwidth. The resulting implementation is a modular design that can be easily adapted to any input instance without the need for complete redesign.

Our solution is asymptotically faster than the state-of-the-art [1]. The mapping time is six orders of magnitude smaller (see Table 1). As a result, the effective speed-up (e.g., considering both the execution and the mapping time) comparing with the solution in [1] is 10^6 . Moreover, the clock speed only depends on the data precision of the input graph instance and not on the size and the connectivity of the graph instance as in [1]. Also, the hardware requirements increase as a linear function of the number of vertices. Consequently, the on-chip execution time and the area requirements can be accurately estimated based on the problem-specific configuration.

Conclusions

In this paper we demonstrated a case-study solution that achieves 6 orders of magnitude speedup over the state-of-the-art for mapping graph-problem instances onto FPGAs. The novelty of our approach is that the mapping process performs an incremental adaptation of problem-specific configurations to the input problem instance instead of a complete redesign. Not only does the approach significantly speedup run-time mapping but also produces fast, compact logic which reduces the execution time on hardware as well.

Our approach can also be applied to other application domains (e.g., image and signal processing, cryptography) where adaptivity to problem instance is required. We believe that by addressing the run-time mapping problem, Reconfigurable Computing can become an attractive computing paradigm for specific military applications.

References

- [1] J. Babb, M. Frank, and A. Agarwal, "Solving graph problems with dynamic computation structures", *SPIE '96: High-Speed Computing, Digital Signal Processing, and Filtering using Reconfigurable Logic*, 1996.
- [2] DARPA ACS Program, <http://www.darpa.mil/ito/research/acs/background.html>