1. Specific Aims

Graphs are central to our understanding of information processing because they are used to describe many disparate types of data including: Electrical and Electronic Circuits, CFGs (Control Flow Graphs) and DFGs (Data Flow Graphs) in Computer Programs, XML Documents, Communication Networks, Molecular and DNA Structures.

Graph mining is a growing area which offers tools and techniques for efficient storage and processing of Graph data. In general, this feat is achieved by identification and encoding common subgraph structures. Thus, graph mining enables the transformation of graph data into knowledge. Owing much to the advances in the various disciplines mentioned in the first paragraph, there has been a tremendous increase in the size as well as the number of datasets/databases comprising graphs. Therefore, we feel that there is a need for practical compression schemes aimed at reducing the byte streams needed to represent graphs in reasonable time.

Acceleration of compute-intensive applications is an active topic of research in HPC (High Performance Computing) community. The philosophy behind this concept is that different parts of an application pose different demands on the execution platform. Therefore, for optimal execution, one has to implement control dominated tasks on general purpose processors and computation intensive tasks on a custom accelerator, and here it is important to decide which instructions should be run on the faster custom hardware. Current work in the field approaches this problem by converting the given program to a DFG. Enumeration techniques are then employed to process this DFG in order to identify subgraphs that give the maximum speed improvement when executed on the accelerators. Various parameters involved with the partitioning of DFG into hardware and software portions include Communication Costs, Computational Complexity, accelerator capability, and execution frequency. The complexity brought about by these parameters render the task of DFG partitioning infeasible for more than a thousand nodes. We would like to develop graph mining tools for enumerating common subgraphs in program DFGs containing more than a thousand nodes as well as finding several subgraphs which together offer speed improvements which potentially exceed that of unique subgraphs.

Circuit minimization is the driving force of the technologies behind the personal computer and the consumer electronics revolutions. While logic minimization has been extensively explored for decades, we propose to pursue explorations targeting FPGAs; we believe there is potential here for parallel computing techniques to find wide application. We plan to devise smart generation techniques to “optimize” small circuits for particular hardware designs and to use these hardware blocks to form larger operations. Oskar Mencer has created an FPGA programming environment called ASC which uses parallelism to facilitate design space exploration. The mathematical component of this project is to understand the underlying symmetries of the problem in order to prune the enumeration search-space. For example, many configurations of lookup tables will have the identical input-output behavior, and these will have to be characterized by shapes and/or latencies. Within a configuration certain inputs will be shared by two or more lookup tables, and this should also reduce the number of possibilities in a smart enumeration. Our main objective is to generate all FPGA circuit blocks with up to 12 lookup tables. The research will have impact if we can identify a few minimal structures that reduce the area and/or latency requirements for frequently used arithmetic functions.

2. Background

For this project we are interested in pursuing three directions where we believe that a better understanding and use of the information content of graphs has the potential to impact multiple disciplines and industries. These are the compression of graphical data, the determination of

critical code segments for hardware-software partitioning of computationally intensive embedded applications and the generation of 'Optimal' small FPGA circuit blocks as a tool for the minimization of layouts for frequently used arithmetic functions.

The literature on graph mining offers tools and techniques to discover important subgraph structures from an input graph. While these structures could be applied toward graph compression, the focus of graph mining research has been on improving the understanding of graph structures and not on practical compression schemes. This latter problem will become increasingly important as more and more graphical data needs to be stored and transmitted.

SUBDUE and GRAPHITOUR are the existing graph mining tools that discuss graph compression. SUBDUE applies a greedy algorithm to seek frequent subgraphs for compressing a graph. Its shortcomings are that it does not offer a compression algorithm that is decompressible, it is known not to discover all of the important subgraphs, and it uses massive computation. GRAPHITOUR considers a single edge type at a time, and may consequently miss more important subgraphs for compression than SUBDUE. Like SUBDUE, its graph grammar may not be decompressible. Neither algorithm considers the actual byte stream needed to represent a graph (see, e.g., [1]).

There is a different graph representation literature which focuses on efficient ways to represent the adjacency list or matrix of a graph, and these ideas have been used in the compressing of web graphs and in finding compact data structures with fast queries for unlabeled graphs. This approach does not consider repeated structures within a graph and therefore seems less suitable for the compression of node-labeled graphs like the ones arising in circuits, in instructions for computations, and in molecular data.

Accelerating Compute Intensive Applications

Previous work in accelerating compute-intensive applications relied on several techniques: heuristic clustering of related DFGs [2, 3], integer linear programming [4, 5], and subgraph enumeration [6, 7, 8, 9]. All of these approaches assumed constraints on the number of input and output operands for the sections to be implemented in hardware (ASIC/FPGAs). However, while the ILP based-approaches are shown to scale relatively well with increasing number of operands [10], enumeration based approaches quickly become intractable with increasing number of operands. In fact recent work [7] showed that the worst case time complexity of enumerating subgraphs having N_{in} inputs and N_{out} outputs in a DFG with N nodes is $O(N^{Nin + Nout + 1})$. Due to these inherent limitations, this work [11] focused only on accelerating applications from the embedded systems domain.

In an attempt to reduce the complexity of the subgraph enumeration problem, Pothineni et al. [12] targeted the maximal convex subgraphs. In essence, Pothineni et al. first define an incompatibility graph in a DFG, where the edges represent pair-wise incompatibilities between DFG nodes. Subsequently, the ancestors and the descendants of an invalid node are deemed incompatible. A node clustering step is introduced for identifying group-wise incompatibilities as well as reducing the size of the incompatibility graph. The incompatibility graph representation allowed Pothineni et al. to formulate the maximal convex subgraph enumeration problem as a maximal independent set enumeration problem thus reducing the complexity of enumeration to $O(2^N)$, where N represents the number of nodes in the incompatibility graph.

Verma et al. [13] used maximal clique enumeration instead of maximal independent set enumeration. However, one can easily transform these two problems into one and other (see for example, Garey and Johnson [14, p.54]). Therefore, the approach of Verma et al. [13] and that of Pothineni et al. [12] are essentially the same.

We propose to develop graph mining techniques suited for much larger programs, possibly from

the High-performance and General-purpose Computing domains, as well as from the embedded systems area, like those present in current literature. We also aim to investigate approaches where multiple subgraphs are ultimately chosen instead of just one and see how this affects the design space.

Circuit Minimization

Circuit minimization has long been the focus of intense research activity for both intellectual and economic reasons. Traditionally, logic synthesis for FPGAs has two phases: technology independent optimization, and technology mapping [15]. The first phase aims to generate an optimal abstract representation of the logic circuit. The second phase tries to transform the abstract representation into a network of lookup tables (LUTs). We limit our attention here to 4-input lookup tables. Many research efforts address the technology mapping problem for LUT-based FPGAs, but it is known [14] that state-of-the-art FPGA technology mapping algorithms miss optimal solutions.

Due to ever-growing computing power, it is now possible to propose a project which would have been out of question even just a few years ago: the brute force generation of all 4-input lookup table-based FPGA circuits of size N for a restricted but growing N.

Brute force generation is a relatively little studied technique for logic minimization which allows us to combine logic minimization and technology mapping into one approach. Generation guarantees that all solutions are considered; one can obtain the absolute lower bound of logic resources needed to implement a particular problem. We are interested in optimization under area and latency constraints. Two recent efforts using enumeration concern an implicit technique for generating structural choices in circuit optimization based on rewiring and resubstitution [14], and the adoption of reverse search in generative optimization for obtaining, for instance, the k shortest Euclidean spanning trees [17].

Our proposed research complements this work - we plan to exploit the large amount of circuit parallelism available in modern FPGAs to speed up the generation process.

3. Significance

Graphs are among the most powerful and flexible forms of data representation because they can model the attributes and relationships among entities in diverse domains ranging from computer science and engineering to biology and chemistry to the social sciences to homeland security. As the number of datasets and databases containing graphical information is growing, there is an increasing need for specialized compression algorithms for graphs offering good compression ratios at reasonable speeds in order to store graph data effectively. Graph mining offer tools and techniques to extract knowledge from graphs. Therefore, any development in this discipline may prove to be an exciting 'enabler' for the advancement of numerous diverse and important disciplines, as described below.

Acceleration of compute-intensive applications is an active topic of research in HPC (High Performance Computing) community. Modern CPUs are well equipped for handling unpredictable patterns in control-flow as well as data. On the other hand, most applications have some compute-intensive, data-parallel code which can benefit immensely from execution on special devices. These devices may include GPUs (Graphics Processing Unit) and FPGAs (Field Programmable Gate Arrays) employed as co-processors along the CPU, as well as application-specific instruction execution units implemented as part of a customizable processor core (ASIPs). With respect to context these devices are called Accelerators. Identification and isolation of sections of code which can benefit from execution on accelerators, followed by their implementation and control is an

involved process. Successful automation of this process has the potential to unlock many wonderful possibilities and therefore is the subject of significant research endeavors.

Microelectronic circuits have been critical for decades to the development of hardware and software systems. Circuit minimization has long been the focus of intense research activity for both intellectual and economic reasons. Companies like Intel and Sony depend on research in logic minimization to develop smaller and cheaper products. The fundamental bounds to minimizing a Boolean function given flexibility at the representation level are central to the foundation of computation. We seek to optimize area and latency of a circuit, either separately or jointly. We propose to use enumeration to (a) find improved bases to implement the behavior of a Boolean function and (b) find small "optimal" building blocks for Field programmable gate arrays (FPGAs). We will also investigate logic functions and sequential circuits with feedback such as state machines. These contributions could lead to significant advances in understanding the intrinsic limitations of digital logic circuits.

FPGAs are widely applied in hardware and software systems in areas such as digital signal processing, aerospace and defense systems, medical imaging, computer vision, speech recognition, cryptography, and bioinformatics. Improvements in logic minimization have the potential to affect industry and ultimately society by guiding the design and improving the effectiveness of future microelectronic circuits. The fundamental bounds to the speed of a computational activity and the minimization of an FPGA circuit are at the foundations of computing, and the proposed activity could lead to significant advances in understanding these topics.

This project will encourage interaction between academia and industry including interdisciplinary and international research experiences for the graduate students involved in the initiative.

4. Preliminary Data or Studies

The researchers initiating this project are: Dr. Serap A. Savari, an expert in data compression and information theory; Dr. Oskar Mencer, a leader in the field of accelerating compute-intensive applications through dedicated accelerators (GPUs, FPGAs, Cell Processor, etc); Dr. Joseph J. Boutros, an experienced researcher in the areas of Coding Theory and Information Theory.

Dr. Savari and Dr. Mencer were coworkers in the Computing Sciences Research Center, Bell Laboratories, and Lucent Technologies. In the last three years they have been collaborating through University of Michigan graduate student Jeehong Yang. Their first project [18] applies logic minimization tools to data compression. Their second project [19] modifies a graph mining tool to obtain an EDIF netlist graph-based compression algorithm that offers the best known compression ratios for these types of files. Netlists contain the connection information of circuits.

Serap Savari has a diverse and interdisciplinary research background. During her undergraduate studies she completed all of the MIT doctoral coursework requirements in Operations Research, and she has a well established track record of applying tools from operations research to problems in information theory and data compression (see, e.g., the standard textbook in information theory, [20, p.713]). With a colleague at Bell Laboratories she applied tools from information theory to a problem in computer architecture in which they compared computer program profiles and created a hybrid profile [21]. This work was recognized by the Best Presentation Prize at the 1999 Workshop on Feedback-Directed Optimization.

She served as an Associate Editor for Source Coding for the IEEE Transactions on Information Theory from 2002-2005 and has been a member of the technical program committee for many conferences and workshops on data compression and information theory. She has two patents in data compression.

Oskar Mencer is a Senior Lecturer in the Computing Department at Imperial College, and he is the

founder of MAXELER Technologies, Inc. He has conducted extensive research on FPGAs (see, e.g., [22] [23], [24]), on computer arithmetic (see, e.g., [25], [26]), and on embedded systems (see, e.g., [27], [28], [29]) His work on solving Boolean satisfiability on FPGAs [30], which is one of the most basic computing algorithms, resulted in over 100x speedup over microprocessors. The same basic approach of iterating over a search space in reconfigurable hardware can be applied to logic minimization. The circuit that is minimized is implemented and explored in the flexible hardware of an FPGA. The key problem is the vast search space for such an activity. His expertise in building parameterized arithmetic units will be critical to accomplishing the work proposed. His PAM-Blox project resulted in a Top Technology article in the EE-Times in 1998. His FPGA programming environment ASC uses parallelism to facilitate design space exploration and will be critical to accomplishing the work proposed.

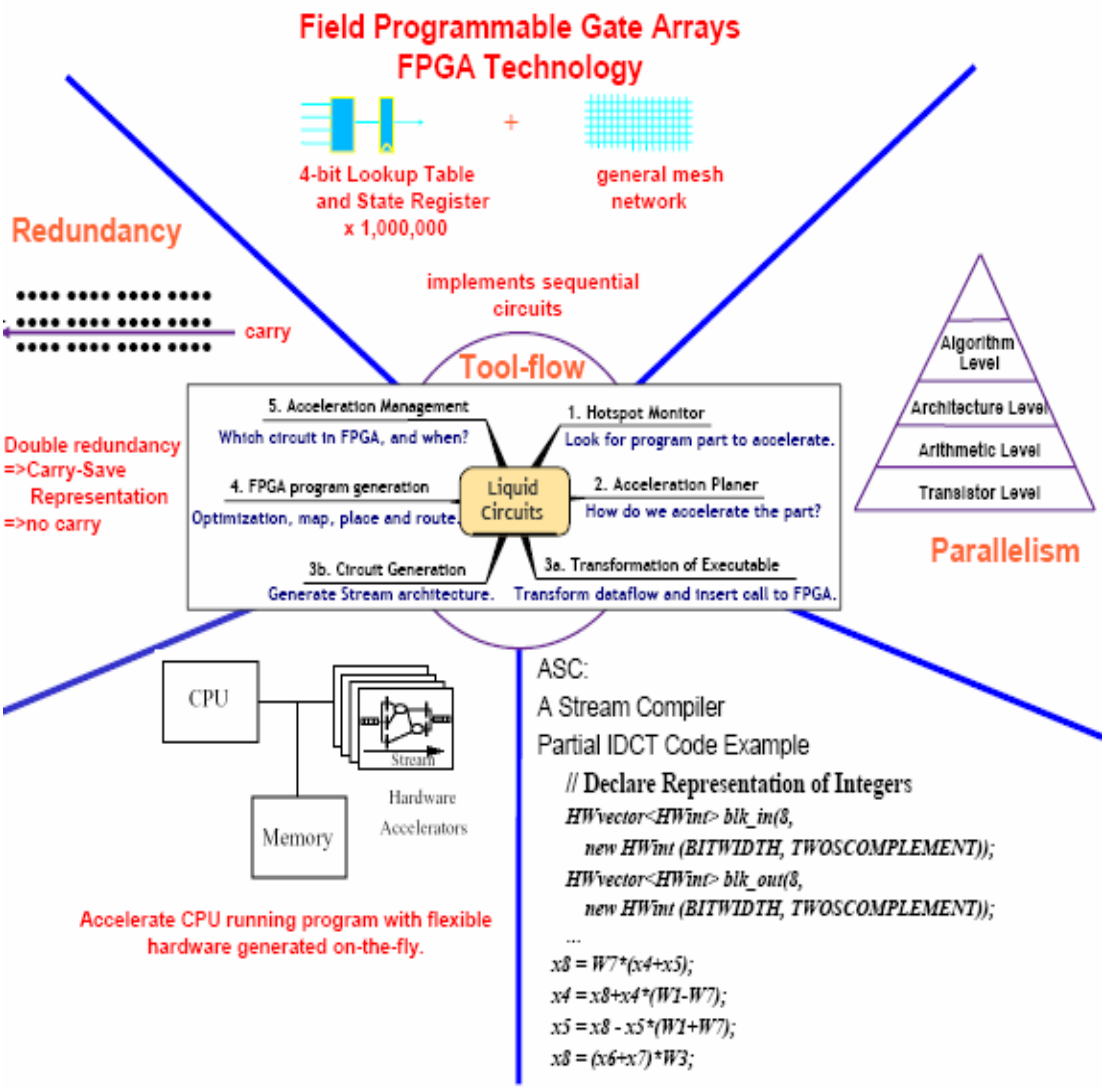


Figure-4.1 *The Liquid Circuit Project being conducted at the Computer Architecture Research Group promises to deliver a PC that can change its circuit depending on the software being executed [36].*

In [30] he explored automatic generation of custom instruction processors from high level programs. The results showed significant speed and area improvements. The work included a novel methodology for identifying custom instructions in a program by enumeration of maximal convex subgraphs of program DFG..

In [31] he presents a methodology for generating optimal architectures for extensible processors. Extensible processors are processors with on-chip FPGA fabric for the implementation of custom instructions. These custom instructions are identified by an ILP (Integer Linear Programming) formulation that extracts the most profitable custom instruction form a given application. Unlike previous approaches to the problem, we differentiate between number of inputs and outputs for the custom instructions and the number of register file ports thus rendering our approach applicable for architectures with visible state registers and dedicated data transfer channels.

In addition he is currently involved with investigating Automated Dynamic Hardware Acceleration for Compute-Intensive Applications - 'The Liquid Circuits Project' at Imperial College, London, Its mission statement reads: “Imagine your PC would change its circuits to match the programs you are running, like a liquid adapting to its container.” [36]. Current research focus is on the identification and isolation of program blocks which can benefit from accelerated execution. Figure-4.1 gives an overall picture of the Liquid Circuits project.

Doctor Joseph J. Boutros received the electrical engineering degree in 1992 and the Ph.D. degree in 1996. From 1996 to 2006, he was with the Communications and Electronics Department at ENST as an Associate Professor. Until 2006 Dr Boutros was a member of the research unit URA-820 of the French National Scientific Research Center (CNRS). In July 2007, Joseph Boutros joined Texas A&M University at Qatar as a Professor in the electrical engineering program.

His fields of interest illustrated in his research, teaching and industrial contracts activities are: Codes on graphs and their iterative decoding including Turbo codes and Low-Density Parity-Check codes, joint source-channel coding for redundant information sources, etc. His research is mainly performed under grants and collaboration with private companies such as Mitsubishi Electric, Motorola Labs, France Telecom, Thales Communications, STMicroelectronics, and Alcatel-Lucent.

At the exception of coding for fading channels with single and multiple antennas where multi-dimensional geometric representations have been used, graph representations and graph-based algorithms were essential in Dr Boutros’ research on trellis codes and low-density codes. The analysis of the code structure and the associated properties such as the Hamming weight distribution is performed via graph theoretical tools (e.g. Boutros and Zemor, IEEE-IT 2006, On Quasi-Cyclic Interleavers for Parallel Turbo Codes). The decoding algorithm for modern graph-based error-correcting codes is also an instance of belief propagation on the local tree-like neighborhood of an information symbol. The decoder’s performance is directly related to the graph structure for both finite and infinite code lengths (e.g. Boutros et.al, IEEE-ISIT 2008, “Generalized Low-density Codes with BCH Constituents for Full-diversity Near-outage Performance”).

The diverse backgrounds and strengths of these researchers uniquely qualify them to undertake this ambitious and interdisciplinary project.

5. Research design and methods

Accelerating Compute Intensive Applications

Implementing computation using accelerators has been shown to be effective for a variety of applications. Compton et al [32] list the speed-ups achieved for implementing various algorithms using reconfigurable accelerators, and [33], details similar advantages of utilizing GPUs for acceleration.

In order to achieve these performance benefits, yet support a wide range of applications, systems are usually formed with a combination of a specialized accelerator and a general-purpose microprocessor. This is due to the fact that there are certain circumstances in which traditional

microprocessors are preferable over the accelerator.

When the function and data granularity to be computed are well-understood and fixed, and when the function can be economically implemented in limited silicon, dedicated hardware (e.g. ASIC or FPGA) provides the most computational capacity per unit area to the application. On the other hand, if we are limited spatially and the function to be computed has a high operation and data diversity, we are forced into reusing limited active space and accepting limited instruction and data bandwidth. In this latter case, conventional microprocessors are most effective since they dedicate considerable space to on-chip instruction storage in order to minimize off-chip instruction traffic while executing descriptively complex tasks. Thus the microprocessor performs the operations that cannot be done efficiently by the accelerator, such as data-dependent control and possibly memory accesses, while the computation intensive portions of the program are mapped to the accelerator hardware.

This accelerator hardware can be composed of either commercial FPGAs, custom reconfigurable logic, specialized functional units within the processor, or even a modern Graphics Processing Unit. Figure-5.1 identifies five possible configurations for combining an accelerator with a CPU. Accelerators that utilize commercially available FPGAs generally fall into category (a). Examples include FPGA based accelerator cards from Nallatech [37] that connect to the system over the PCI Bus, and do not have direct access to the system's main memory. Recently, with the advent of Hypertransport, companies such as DRC [38] have produced devices that communicated directly with the CPU, as well as its Main Memory (b). Custom designed Reconfigurable units such as GARP, and modern ASIPs tend to fall into categories (c) and (d) respectively. Examples of category (e) systems are high-end FPGAs from Xilinx, Altera etc that incorporate a general-purpose CPU into their reconfigurable fabric, either as a soft-processor core, or as a dedicated hardware block.

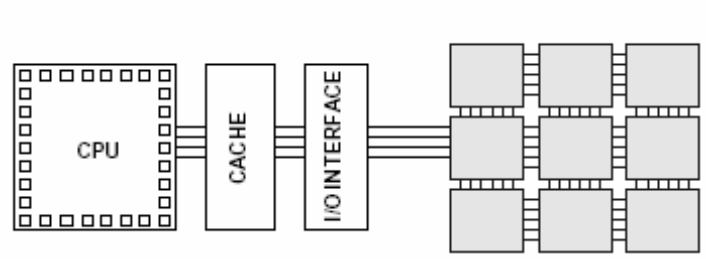
Accelerating Applications Using FPGAs

It can be stated that the implementation of computational algorithms in configurable devices such as FPGAs is accomplished in 'Space Domain'. On the other hand, the implementation of computational algorithms in microprocessors is executed in 'Time Domain'. Unlike the microprocessors which broadcast instructions to the functional units on every clock cycle, instructions in reconfigurable devices are locally configured, allowing the reconfigurable device to compress instruction stream distribution and thus deliver more instructions into active silicon on each cycle.

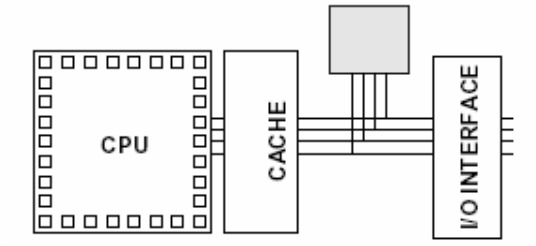
Reconfigurable devices provide a large number of separately programmable, relatively small computational units allowing for the execution of a greater range of computations per unit time. Resources such as memories, crossbar switch matrices and logic blocks are distributed rather than being centralized in large pools. Independent, local access allows for the utilization of on-chip resources in parallel, thus avoiding performance bottlenecks resulting from a large, central resource pool. Therefore, it can be stated that the control portions of the algorithm call for microprocessor implementation while the computationally expensive 'performance critical' portions of the algorithm should benefit from the custom hardware options provided by FPGA.

Application Specific Instruction-Set Processors

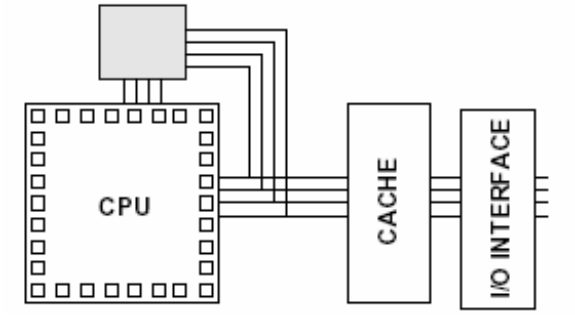
Embedded processors carrying out special purpose computations currently make up the majority of sales in the microprocessor market and are critical to the consumer electronics, multimedia, and communications industries. Annually over a billion embedded processors are components of new embedded devices. Because of this high volume, embedded systems are built under stringent cost and performance constraints, and there is a continual drive for new products with improved functionality and efficiency.



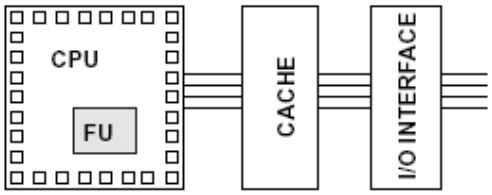
(a) External Stand-alone Processing Unit



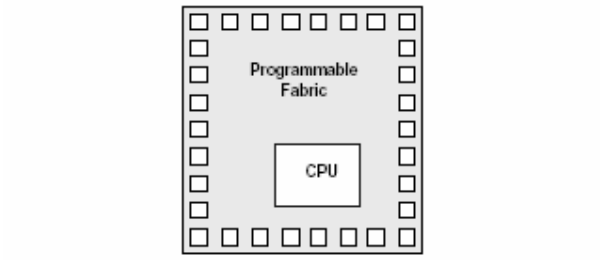
(b) Attached Processing Unit



(c) Co-processor



(d) Functional Unit



(e) Processor embedded in reconfigurable fabric

Figure-5.1 *Configurations for combining an accelerator with a general-purpose CPU.*

One of the major challenges faced by producers of embedded systems-on-chip devices is the inability of designers to keep apace with the ever-increasing demands in logic complexity [34]. Furthermore, computer programs have been observed to spend about 90% of their execution times in about 10% of the code [35]. These phenomena have led to a paradigm where there is an attempt to implement the most computation intensive parts of the code on application-specific integrated circuits, field programmable gate arrays (FPGAs), or to utilize ASIPs – customizable processing cores that are able to incorporate special-purpose instructions specifically tailored to the relevant application domain. The architectural model we consider builds upon a pre-verified and pre-optimized base processor with a basic instruction set. This processor is supported by custom

functional units that implement application-specific instructions; these instructions are sometimes called custom instructions, and become a part of the CPU pipeline, much like the ALU. Unlike the ALU, however, they implement as dedicated hardware blocks the frequently executed, computationally intensive operations that are extensively utilized by the targeted application or application domain. This is illustrated in Figure-5.2.

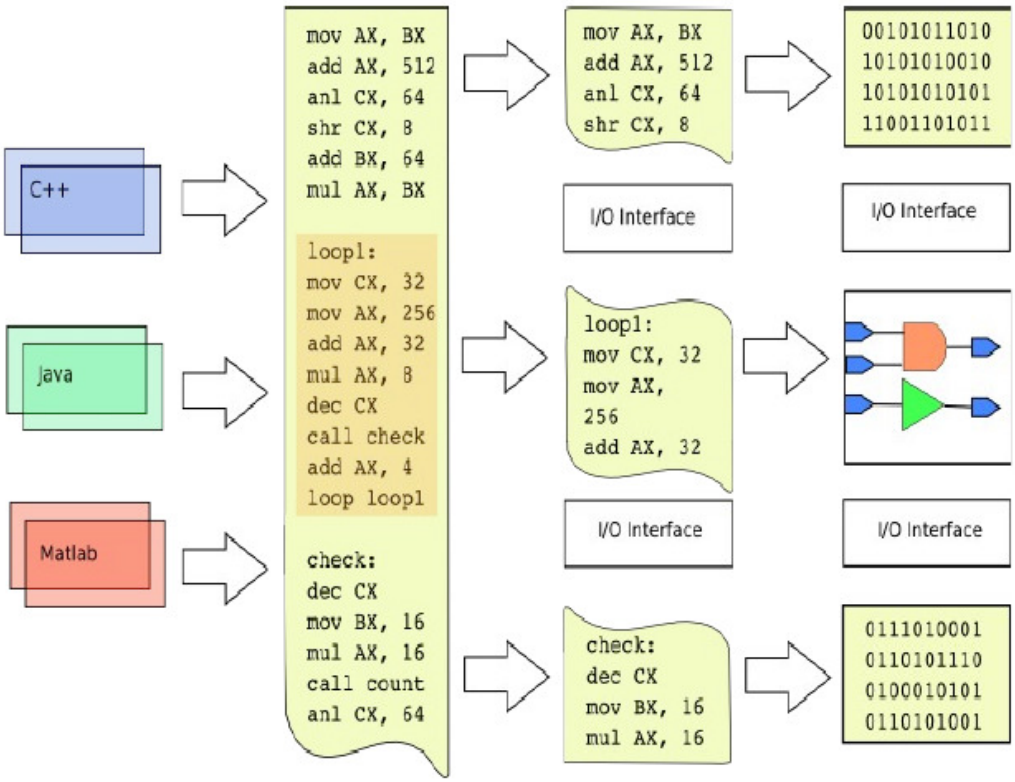


Figure-5.2 *An illustration of the extraction of performance critical program blocks form software and their implementation on hardware platform. Notice that the source code can be in any programming language. This is possible because the program blocks are being extracted at the assembly level. Notice that the execution time of the final implementation depends upon the bandwidth of the I/O interface.*

Although this model has received attention from the academic literature and from industry, there is limited understanding of effective ways to choose which instructions should be executed by the base processor and which should be carried out by the custom hardware. This is partly because the partitioning is constrained by a number of factors including limitations on custom logic area, on data transfer costs between the base processor and the custom units, on the structure of the custom instructions, and on power consumption.

Accelerating Applications Using GPUs

The past few years have seen the transformation of the GPU (Graphic Processing Unit) from a chip intended to support the CPU for specialized tasks such as 3D and texture rendering etc. into a general purpose computational powerhouse. This meteoric rise in computational power is evident from Figure-5.3. This computational power is delivered by multiple cores with access to high memory bandwidth with an emphasis on data processing rather than data caching and flow control. This distinctive architecture was intended for the compute intensive, highly parallel graphic

rendering applications.

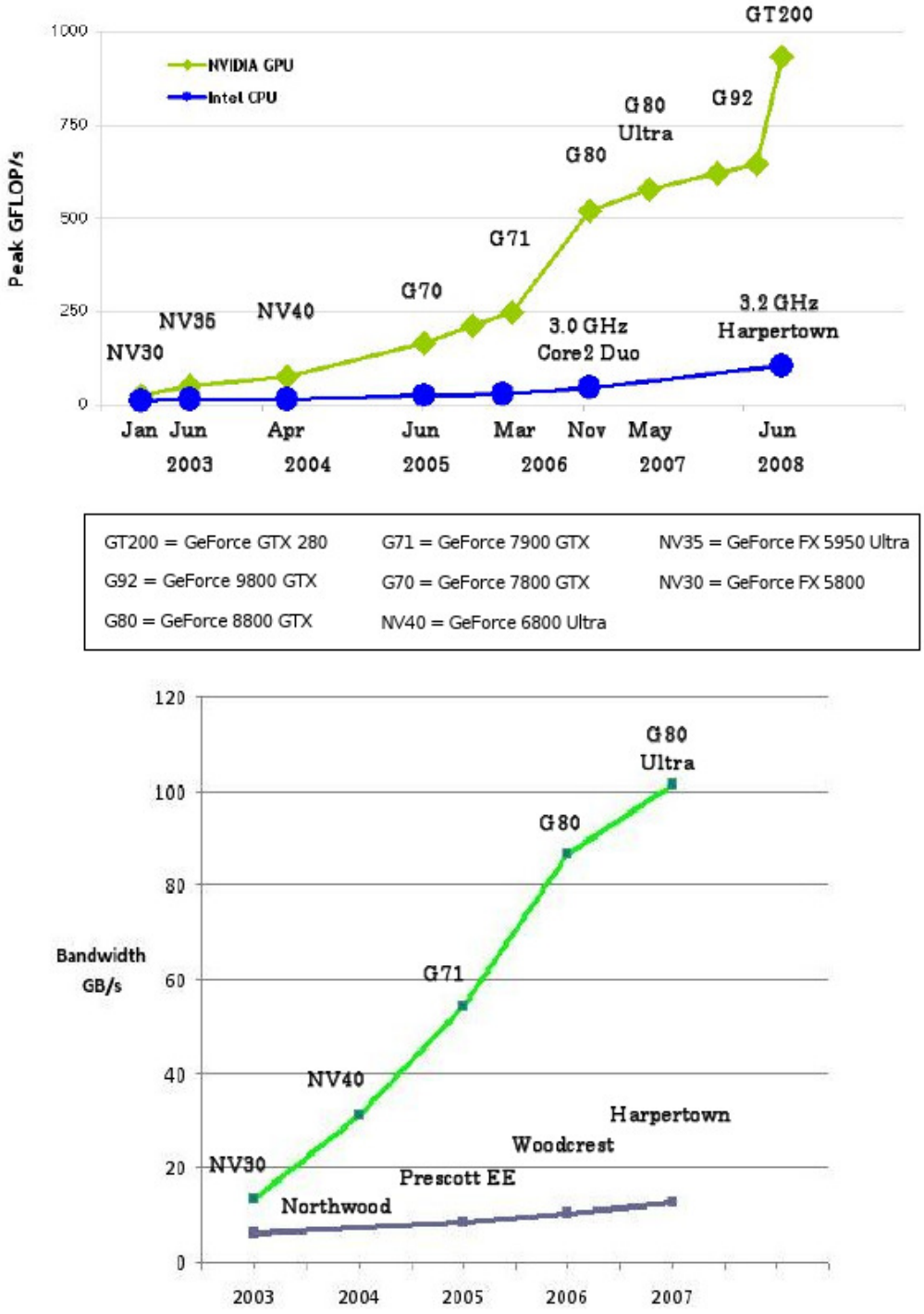


Figure-5.3 *FPU (Floating-point Operations) per Second for CPUs and GPUs [3]. The growth in computational power of CPUs is considered meteoric, however it dwarfs in comparison to the growth of GPUs.*

More specifically, the GPU is well-suited to SIMD (Single Instruction Multiple Data) applications in which the same program is executed on multiple data elements in parallel. Usually the program

performs operations which have high arithmetic intensity i.e. ratio of arithmetic to memory access and control instructions. Therefore, there is less demand for sophisticated instruction and data caching, flow control and branch prediction mechanisms. The precious silicon resources can therefore be utilized to provide a much higher computational throughput. This throughput is usually delivered by multiple execution cores enabling simultaneous processing of multiple data elements.

Many applications that process large data sets such as arrays can use a data parallel programming model to speed up the computations. In 3D rendering large sets of pixels and vertices are mapped to parallel threads. Similarly, image and media processing applications such as post-processing of rendered images, video encoding and decoding, image scaling, stereo vision, and pattern recognition can map image blocks and pixels to parallel processing threads. In fact, many algorithms outside the field of image rendering and processing are accelerated by data-parallel processing, from general signal processing or physics simulation to computational finance or computational biology.

Examples Applications using GPUs/Reconfigurable Logic

The Table below summarizes some of the efforts to enhance the performance of various applications using accelerators.

Application	Accelerator	Results
Cryptography – Serpent Block Cipher	Implemented on Xilinx Virtex XCV1000 FPGA	Throughput increase by a factor of 18 over a general purpose Pentium Pro Processor [32, 46].
Cryptography - DES	Implemented on the custom GARP reconfigurable functional unit	Order of magnitude increase in throughput over a CPU only implementation [32, 47].
Search for Prime Numbers	Implemented using the Custom Mojave Configurable Computing Architecture	Accelerated by a factor of 28 over an Ultrasparc Processor [32, 48].
Generalized Sparse Linear Solver	Implemented on ATI Radeon X1900XTX, and Nvidia Quadro FX 5600 GPUs, and an Athlon 64 4800+ X2 CPU	Individual benchmarks 16 times faster, and the linear Solver accelerated by 6 times over the CPU [49].
Quantum Chemistry – Two Electron Integral Evaluation	Implemented on GPU and an AMD Opteron CPU.	Achieved 160 fold speedup over AMD Opteron CPU [50]

In addition, recently several researchers and corporations have demonstrated significant improvements through the utilization of accelerators, in application areas as diverse as Computer Aided Engineering, Computational Fluid Dynamics, Reservoir Simulation (for the petroleum industry), Computational Chemistry and Physics, Cryptography and Compression, Real-time Image and Video Processing, etc. Additional details and examples of utilizing GPUs and FPGAs for acceleration may be found at [51, 52].

Proposed Research

In order to achieve significant speed-ups for any high-performance or general-purpose computing applications using a dedicated accelerator, the application needs to be efficiently partitioned into

sections of code that will run on a conventional CPU, and sections of code that can be effectively be implemented on the accelerator hardware. This partitioning requires that the application code be first transformed into a compiler intermediate representation, usually in graphical form – i.e. control or data-flow graphs. Then, sub-graph enumeration techniques are used to identify one or more profitable sub-graphs that would benefit from being implemented on the custom accelerator hardware.

The state-of-the-art along these lines handles graphs with about a thousand nodes. While this is usually sufficient for small applications, such as those in the embedded-systems domain, these graph mining tools are not suitable for extracting common sub-graphs from the much larger graphs that would represent the larger, more complex applications from the areas of high-performance or general-purpose computation. We propose:

- To develop graph mining techniques that are well suited to analyze much larger program graphs and more efficiently identify profitable sub-graphs. Our research will have to account for the differing requirements of the various types of accelerators. For instance, an FPGA or GPU would more efficiently implement medium to large sections of code that execute for long durations and exploit large amounts of data parallelism. On the other hand, the custom-instruction elements for ASIPs would instead require the identification of small, frequently executed sections of code, that only run for tens of clock cycles at most.
- To investigate approaches where multiple sub-graphs are ultimately chosen instead of just one and see how this affects the design space; this would be somewhat analogous to the graph covering approach for the compression portion of the project.

In order to properly understand the problem space and develop possible solutions, our proposed methodology for exploring the acceleration of compute intensive applications is as follows.

1. Study and analyze the source code of common general-purpose and high-performance computing application benchmarks, such as the SPEC suite [39], as well as the 13 Berkeley Dwarfs [40].
2. Use a basic compiler infrastructure such as SUIF [41] to extract the intermediate representation graphs of these benchmarks.
3. Utilize these Control/Data-flow graphs as the test inputs for existing graph mining tools, as well as our own, enhanced graph mining algorithms. This would allow us to extract the appropriate sub-graphs for implementation on different types of accelerators.
4. Compile both sets of intermediate representations and implement them on the appropriate computation platforms – CPU+GPU, CPU+FPGA, ASIP, etc.
5. By comparing the performance of the benchmarks implemented on a specific accelerated platform using our graph mining algorithms with the performance extracted from the same platform running the same benchmarks, but using the existing graph mining tools instead, we can evaluate the effectiveness of our modifications.

Logic Minimization

Due to ever-growing computing power, it is now possible to propose a project which would have been out of question even just a few years ago: the brute force generation of all 4-input lookup table-based FPGA circuits of size N for a restricted but growing N. For example, the truth table for an N-bit input, 1-bit output function Y is given in Table 1.

Brute force generation is a relatively little studied technique for logic minimization which allows us to combine logic minimization and technology mapping into one approach. Generation

guarantees that all solutions are considered; one can obtain the absolute lower bound of logic resources needed to implement a particular problem. We are interested in optimization under area and latency constraints. Table 2 begins to illustrate the range of the design space for single output functions. Two recent efforts using enumeration concern an implicit technique for generating structural choices in circuit optimization based on rewiring and resubstitution [42], and the adoption of reverse search in generative optimization for obtaining, for instance, the k shortest Euclidean spanning trees [43].

Our proposed research complements this work since we plan to exploit circuit parallelism in speeding up the generation process. The successful implementation of this proposal has the potential to change the standard way of thinking about hardware design and could lead to major advances in industry. With silicon area increasing exponentially, this method will become more and more feasible over time. For one 4-input lookup table, generation is trivial, i.e. there are 2^{16} possible Boolean functions stored in the LUT. Connecting four bits to a LUT is easy because it does not matter which pin is connected to which bit since the LUT content includes permutations of inputs. However, if we want to explore designs with 6 LUTs we have roughly $2^{6*16} = 2^{96}$ possible states, which is already quite large. However, using FPGAs to search the design space, we will soon be able to fit many millions of (6 LUT) blocks into one FPGA running at hundreds of MHz, and thus we can explore more than 2^{50} design points per second per FPGA. Of course the actual algorithm is slightly more involved than this description. By carefully eliminating redundant cases within the generation space we expect to increase the number of LUTs we can generate to over 12.

The overall goal of this part of the research program is to find optimal circuits for a particular hardware design by smart generation. Smart generation enables us to generate the design space in a particular order to minimize the time to find a solution. We plan to accomplish this task by first generating all possible circuits up to a particular size and then combining these hardware blocks to form larger operations. Once we have a way to find optimal circuits with a few LUTs, we partition larger functions based on regularity found within them. The simplest such functions of interest are multiplication, division, and square-root. An understanding of such regularity, coupled with optimization methods such as genetic algorithms and simulated annealing will lead us to complete implementations of arithmetic functions in hardware using small “optimal” building blocks.

This research project will be guided by O. Mencer’s experience in developing parameterized FPGA libraries in C++. C++ is a versatile and powerful language that will simplify the seemingly enormous generation task outlined above. His FPGA programming environment, A Stream Compiler – ASC – [31] enables rapid design-space exploration and has a single representation of the whole software- hardware system. We propose to use ASC as the underlying FPGA programming environment. ASC provides a software-like interface to programming hardware accelerators such as FPGAs while staying competitive with hand-designed circuits on the gate-level. ASC accomplishes this task by providing a user interface that enables programming on the algorithm level, the architecture level, the arithmetic unit level and the bit level, all within a single C++ program. Thus, ASC enables the programmer to exploit parallelism on all levels from algorithm level all the way down to the bit level. In essence, ASC is a C++ library and, as such, can be compiled by a standard C++ compiler. Thus, ASC code is simply C++ which makes use of the ASC library in a compliant manner. When compiled, ASC code becomes an executable which either acts as a word level simulation, a bit-level simulation, or produces a circuit in the form of a hardware netlist. In order to express and explore the design space of a hardware accelerator, ASC code is parameterized to generate a large selection of implementations. With these parameterizations the user trades off, for example, silicon area for latency, throughput, and/or precision.

The generation search space is highly symmetric – the same subsearch occurs many times.

Symmetries occur at many levels. For example, for two N-input designs, the same graphs and LUT configurations will be generated. The emulation of a LUT with one configuration-input pair is equivalent to many configuration-input pairs; the same output results. We propose to adapt O. Mencer's parallel generation hardware to take advantage of these symmetries. The research can be considered successful if we can identify a few minimal structures that reduce the area requirement subject to latency constraints for frequently used arithmetic functions

Proposed Research

In this part of the proposal, we address the problem of optimization of designs for reconfigurable hardware. We use enumeration to optimize FPGAs under area and latency constraints. We only consider combinatorial designs, including the state-transition logic of finite state machines. To simplify the discussion here, we further initially restrict our attention to single outputs (no sharing of logic between multiple outputs), but we plan to extend our ideas to multi-output functions.

A first approach to enumeration is to break up the problem into several steps:

Step-1 Given a Boolean input function Y and an optimization metric (area or latency), identify observable inputs and limit the search space.

Step-2 Enumerate all circuit shapes within the search space from Step 1, sort by (a) latency or (b) area,

Step-3 Enumerate all possible interconnections for each shape,

Step-4 Enumerate all possible LUT configurations for each circuit.

Steps 1, 2 and 3 can be implemented in software. Step 4 can be implemented by parallel enumeration using FPGAs. Hardware enumeration using FPGAs relies on two key properties of these devices:

LUTs: FPGAs can implement high-speed table lookup. Restricting the design space for enumeration to consider only LUTs, we reduce our problem to finding LUT configurations and interconnections.

Massive Parallelism: We can run many instances of our enumeration designs in parallel, with a commensurate speedup.

Expressions for Upper and Lower Bounds

In this section we develop expressions for the upper bound of the design space for enumeration. We assume layers of LUTs to realize a design, for which we enumerate different LUT configurations and interconnections.

0	1	...	i	...	N-1	Y
0	0	0	0	0	0	y_0
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$c_{0,t}$	$c_{1,t}$	...	$c_{i,t}$	...	$c_{N-1,t}$	y_t
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	1	1	1	y_{2^N-1}

Table-1 *Truth table for a Boolean function with N design inputs and one design output.*

We assume that the truth table for an N-bit input, 1-bit output function Y is given as in Table-1. We assume that the function has already been reduced so all inputs are observable; observability of an input can be computed using Boolean derivatives as defined in [14]. We enumerate designs consisting of 4-input LUTs. We assume that the circuit is composed of H layers of LUTs, where layer h is composed of L_h LUTs.

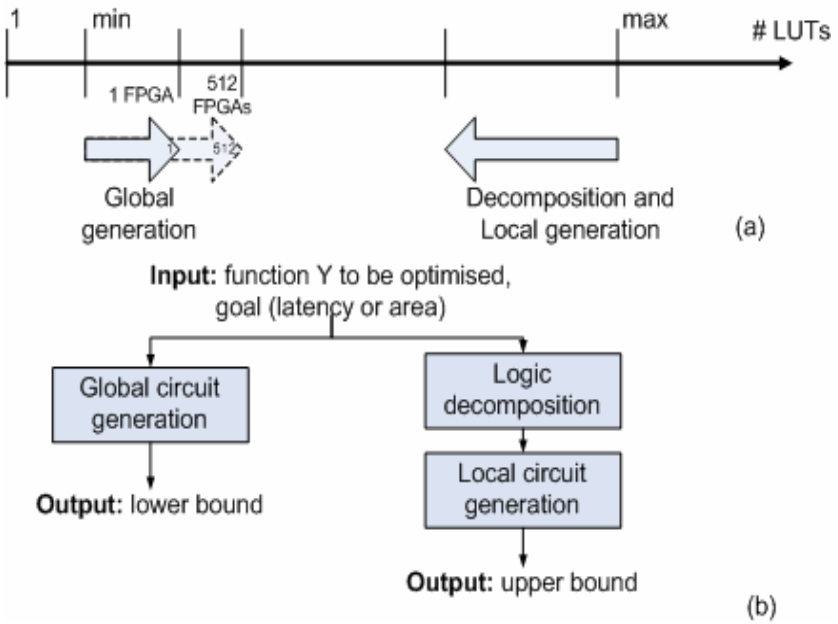


Figure-5.4 *We find the lower bound using global generation while the upper bound is found using logic decomposition.*

We define L_{tot} as the total number of LUTs contained in the design. We refine our 4-step enumeration process to handle N-input logic functions:

Step-1

We identify observable inputs and index into Table-2 to find the range of our design space with associated area and latency requirements. For the purposes of this proposal, we define latency as the maximal depth in LUTs from design inputs to design output, and area as the total number of LUTs in a design. The maximum latency and area requirements are calculated based on the following observations:

- A 4-input design can be implemented by a single LUT,
- An (n + 1)-input design can be implemented using two n-input designs, and an additional LUT multiplexing between the two using the (n + 1)th input.

The minimum area and latency requirements are calculated based on the following rules:

- Each observable design input must be connected to at least one LUT input,
- At least one of the LUT inputs must be connected to a LUT output at a previous layer, there is a single LUT at the highest layer.

Fig. 4 illustrates these requirements. These rules ensure that (1) no input is redundant, (2) no LUT is disconnected (redundant) and (3) there is only one design output.

Definition-1: The shape of a connection of LUTs is a vector, with each element of the vector being the number of LUTs in that layer.

function	optimize for latency		optimize for area	
#inputs	min	max	min	max
≤ 4	1	1	1	1
5	2	2	2	3
6	2	3	2	7
7	2	4	2	15
8	2	5	3	31
9	2	6	3	63
10	2	7	3	127
11	2	8	4	255
N	$log_4(N)$	$(N-3)$	$\lfloor (N+1)/3 \rfloor$	$2^{N-3}-1$
	$O(logN)$	$O(N)$	$O(N)$	$O(2^N)$

Table-2 *Latency (maximum number of LUTs from inputs to output) and area (number of LUTs) for differing numbers of inputs. User input to optimization is # inputs and optimization mode (latency or area). (Step 1)*

Step-2

We find all shapes for the range found in Step 1 (Table 3). We sort the resulting list of shapes by latency or area. For example, to enumerate an 8-input design for minimum area, we first choose the smallest shape that will accept eight inputs: (2,1) in our terminology. If this fails, we choose one of the next smallest designs, and so on. Similarly, the minimum latency design can be found by iterating from the minimum latency topology to the maximum.

	Latency				
	1	2	3	4	5
Area					
1	(1)				
2		(1,1)			
3		(2,1)	(1,1,1)		
4		(3,1)	(2,1,1)	(1,1,1,1)	
			(1,2,1)		
5		(4,1)	(3,1,1)	(2,1,1,1)	(1,1,1,1,1)
			(1,3,1)	(1,2,1,1)	
			(2,2,1)	(1,1,2,1)	
6			(4,1,1)	(3,1,1,1)	(2,1,1,1,1)
			(3,2,1)	(2,2,1,1)	(1,2,1,1,1)
			(2,3,1)	(1,3,1,1)	(1,1,2,1,1)
			(1,4,1)	(2,1,2,1)	(1,1,1,2,1)
				(1,2,2,1)	
				(1,1,3,1)	

Table-3 *All the different shapes for one to four 4-LUTs, arranged according to latency and area. (Step 2)*

We observe that the number of shapes for a given number of LUTs L_{tot} and layers H is bounded by the binomial coefficient $\binom{L_{tot}}{H}$. Thus the total number of shapes is bounded by $2^{L_{tot}}$ [44], and the total number of shapes for the range of areas from step 1 is bounded by:

$$\sum_{\lfloor (N+1)/3 \rfloor}^{2^{N-3}-1} 2^{L_{tot}} = 2^{2^{N-3}} - 2^{\lfloor (N+1)/3 \rfloor}$$

Step-3

We enumerate all interconnection possibilities. In Step 3(A), we produce a set of connections for each shape: topologically distinct trees where the output of each LUT in a layer must connect to the input of a LUT in the next layer. For example, Fig. 5 shows the two possible connections for the shape (2,2,1). Step 3(B) then enumerates directed acyclic graphs for each connection, producing all combinations of connections from each LUT input unconnected in Step 3(A) to each LUT output in previous layers, and the design inputs. For a LUT at layer h, the number of possible interconnections is:

$$L_{h-1} * (N + \textstyle\sum_0^{h-1} L_i)^3.$$

Step-4

For all graphs, we enumerate each configuration of each LUT, for each input. For L_{tot} LUTs, there 4 are $2^{(2\text{^}4)*Ltot}$ configurations. The output of the final circuit must be identical to the N-bit function output specified by a truth table for each input over the input space of 2 N (See Table 1).

To simplify our discussion here, we only consider combinatorial designs (no registers or feedback) with single output. Enumeration can still apply to the combinatorial parts of sequential designs, and we propose to address multiple-output designs by relaxing the requirement that there is a single LUT at the highest layer. Sequential designs present another challenge, requiring us to consider time as well as space, but allow us to subsume retiming into a single optimization step with logic optimization and technology mapping. A good approach might be to consider important subsets of sequential designs, such as pipelined designs, or loops with simple dependencies. The enumeration approach outlined above is basic, and it would take a long time even for simple circuits. In order to accelerate progress we will need to understand the underlying symmetries of the problem in order to prune the enumeration search-space. We discuss this in more detail as we next propose our approach to parallel hardware enumeration, but we point out here that smart enumeration for combinatorial optimization problems is generally nontrivial [45].

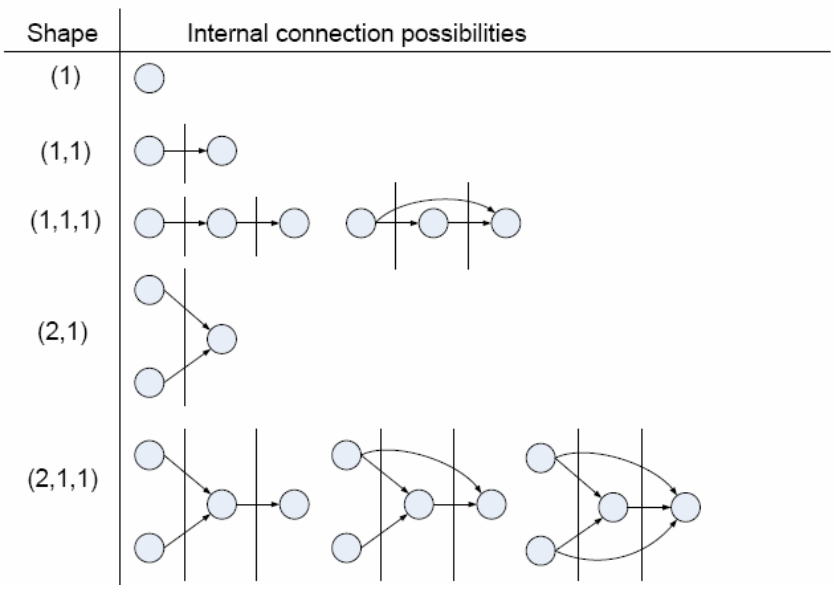


Figure-4 *Some internal connection possibilities for several shapes. Vertical lines separate the layers. (Step 3)*

Figure-5 *Step 3(A): generating the two connections for the shape (2,2,1) – the two topologically distinct trees formed by connecting each LUT output in a layer to a LUT input in the next layer.*

Parallel Hardware Enumeration: This research project will be guided by O. Mencer's experience in developing parameterized FPGA libraries in C++. C++ is a versatile and powerful language that will simplify the seemingly enormous task outlined in the previous section. His FPGA programming environment, A Stream Compiler – ASC – [31] enables rapid design-space exploration and has a single representation of the whole software-hardware system.

We propose to use ASC as the underlying FPGA programming environment, depicted in Fig. 6. ASC, A Stream Compiler, provides a software-like interface to programming hardware accelerators such as FPGAs while staying competitive with hand-designed circuits on the gate-level. ASC accomplishes this task by providing a user interface that enables programming on the algorithm level, the architecture level, the arithmetic unit level and the bit level, all within a single C++ program. Thus, ASC enables the programmer to exploit parallelism on all levels from algorithm level all the way down to the bit level. In essence, ASC is a C++ library and, as such, can be compiled by a standard C++ compiler. Thus, ASC code is simply C++ which makes use of the ASC library in a compliant manner. When compiled, ASC code becomes an executable which either acts as a word level simulation, a bit-level simulation, or produces a circuit in the form of a hardware netlist. In order to express and explore the design space of a hardware accelerator, ASC code is parameterized to generate a large selection of implementations. With these parameterizations the user trades off, for example, silicon area for latency, throughput, and/or precision.

The enumeration search space is highly symmetric – the same subsearch occurs many times. Symmetries occur at many levels:

Inter-design for two N-input designs, the same graphs and configurations will be enumerated.

Inter-graph for a single design, many of the graphs produced in Step 2 will share combinations of inputs and LUT configurations.

Inter-LUT in a single graph, LUTs will share inputs and configurations.

Intra-LUT emulation of a LUT with one configuration-input pair is equivalent to many other configuration input pairs; the same output results.

We propose to adapt O. Mencer's parallel enumeration hardware to take advantage of these symmetries. Using ASC to implement the enumeration of circuits, our main objectives are:

- To design a scalable hardware structure for the enumeration of FPGA circuits based on 4-input lookup tables (LUTs).
- To enumerate all FPGA circuit blocks with up to 12 LUTs by:
  - Enumerating all circuits of depth 1 LUT, 2 LUTs, 3 LUTs, etc.
  - Enumerating all ways of interconnecting the LUTs.

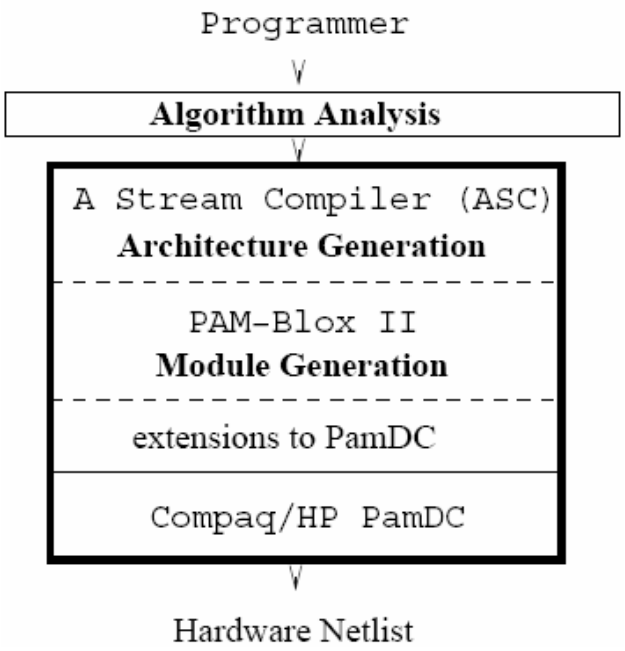


Figure-6 *Levels of abstraction and structure of ASC. The bold box represents a single C++ program.*

The research can be considered successful if we can identify a few minimal structures that reduce the area requirement subject to latency constraints for frequently used arithmetic functions.

6. Anticipated Results and Evaluation criteria

Acceleration of compute-intensive applications is an active topic of research in HPC (High Performance Computing) community. In general these applications are categorized on the bases of their target area e.g. Embedded computing, Scientific computing, Multimedia and signal processing etc.

Several benchmarks are available in these areas for verification of research results and to act as testing criteria. For instance The Standard Performance Evaluation Corporation (SPEC) is a non-profit corporation formed to establish, maintain and endorse a standardized set of relevant

benchmarks that can be applied to the newest generation of high-performance computers. SPEC includes dedicated benchmark suites for General purpose microprocessors, Graphics processors and Workstations, MPI/OMP based parallel computers, Java Client/Server systems, Mail Servers, Network File Systems and Web Servers. Similarly MiBench is a benchmark suite from University of Michigan at Ann Arbor comprising common representative applications from the embedded systems domain. It includes numerous applications distributed into categories such as automotive, consumer, network, office, security and telecommunications.

We aim to employ graph mining tools for enumerating common subgraphs in program DFGs containing more than a thousand nodes thus targeting a spectrum of applications starting from large embedded programs to scientific simulations. We will employ enumeration techniques are then employed to process this DFG in order to identify subgraphs that give the maximum speed improvement when executed on the accelerators. Various parameters involved with the partitioning of DFG into hardware and software portions include Communication Costs, Computational Complexity, accelerator capability, and execution frequency. The complexity brought about by these parameters render the task of DFG partitioning infeasible for more than a thousand nodes. We would like to develop graph mining tools for enumerating common subgraphs in program DFGs containing more than a thousand nodes as well as finding several subgraphs which together offer speed improvements which potentially exceed that of unique subgraphs.

Specifically our efforts would result in a framework based on graph mining tools which given a program will analyze it in order to find common patterns such as compute intensive loops. Subsequently, these common patterns will be implemented to execute on a specialized accelerator therefore accelerating the program.

A significant breakthrough in accelerating the execution of compute intensive programs will be automated employment of FPGAs as accelerators. Circuit minimization will be the driving force behind this revolution. It will enable us to implement complex arithmetic circuits in minimal hardware thus guaranteeing optimal occupation of precious silicon resources in the FPGA fabric. While logic minimization has been extensively explored for decades, we propose to pursue explorations targeting FPGAs; we believe there is potential here for parallel computing techniques to find wide application.

We plan to devise smart generation techniques to “optimize” small circuits for particular hardware designs and to use these hardware blocks to form larger operations. Our main objective is to generate all FPGA circuit blocks with up to 12 lookup tables. The research will have impact if we can identify a few minimal structures that reduce the area and/or latency requirements for frequently used arithmetic functions.

7. Plans for disseminating research results

We will continue our record of widely disseminating technical results by means of journal publications, presentations at conferences and workshops and visits to other universities. The student(s) supported by this grant will participate actively in relevant national and international professional meetings such as FCCM and FPL.

The student(s) sponsored by this award will continue in the path of Jeehong Yang and participate in visits to Imperial College. Oskar Mencer has mentored many graduate students through academia and industrial internships, and these experiences generated publications for the students.

S. Savari was involved with educational outreach programs designed to support high school and undergraduate university students who are pursuing careers in information and communication technologies. She participated in many activities designed to make science and engineering more interesting and accessible to the general public as well as in programs designed to increase the

representation of women in these disciplines. At the University of Michigan, Ann Arbor she advised the group GEECS, Girls in Electrical Engineering and Computer Science, which provides a social and academic network for undergraduate and graduate women students in her department. We will seek opportunities to engage in similar projects.

Dr Boutros will take a special care in disseminating the scientific results of this project within the coding and communications communities by the means of workshops and visits to worldwide-known research and academic laboratories. Indeed, the acceleration of many computational procedures used in coding systems should greatly benefit from our results (e.g. implementation of an accelerated Sphere Decoder or fast Density Evolution). Indirect dissemination can be done via senior design projects at TAMUQ in Doha. Direct local dissemination is done via scientific conferences in the Gulf region around Qatar. Research results will be published in high-impact international journals and presented at leading conferences in the field. In particular, we will target the IEEE Transactions on Information Theory, ACM Transactions on Reconfigurable Technology and Systems, and the conferences IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM) and IEEE International Conference on Field-Programmable Technology (FPT). In order to allow for a rapid dissemination of our most recent results, we will post the submitted manuscripts on arXiv (http://arxiv.org/). Moreover, in order to reach a broader audience, selected results will be published in more general-scope conferences such as the IEEE Computer Magazine, and IEEE Transactions on Computers, etc.

As part of the research program, we will pursue technology transfer opportunities for the results from our work. Public seminars and colloquia will be organized, targeting participation by industrial and other research organizations. Ongoing research results will be made available over the World-Wide Web – our plan includes the development of a project website to elaborate our project goals and objectives, facilitate the dissemination of our results, as well as encourage greater collaboration with researchers across the world interested in our work. Interaction with national and international industry forums will be used to discuss the practical implications of the proposed research in future commercial systems.

In addition, the principal researchers will continue our professional activities such as serving on technical program committees of major conferences and workshops, performing editorial work, and organizing professional meetings.

8. Strategy for project continuation

The PIs have established solid research collaboration since for the past several years, and this collaboration will continue after the completion of this project.

One of the primary outcomes of this project would be the successful utilization of efficient graph mining algorithms to extract the computationally intensive sections of an application source code, for mapping onto accelerator hardware. Our work in logic minimization through enumeration would also enable us to map these sections with near-optimal efficiency onto modern FPGAs. The success of these endeavors would immediately open up further research opportunities in two distinct areas:

1. In order to maximize the impact of this outcome, these graph mining algorithms would have to be automated and incorporated into an efficient high-level language compiler framework, such as gcc, or suif. Additional research in this direction will focus on automatically compiling code written in high-level languages for computing systems that incorporate different types of accelerators along with their CPUs. The type, capabilities, constraints, and other parameters of each supported accelerator would have to be taken into consideration when developing the compiler framework, so that the code partitioning is close to optimal for each type.

2. While a compiler incorporating efficient graph mining algorithms would be suitable for use with newly written applications, or those whose source code is available, the vast majority of legacy applications would be unable to benefit from this effort, since their source code is either proprietary, or no longer available. In order to provide accelerator support to such applications, another area of future research would be to explore the analysis and acceleration of these applications at runtime. It may be possible to utilize runtime analysis/profiling tools to generate the CFG/DFGs for the code at runtime. Utilizing our enhanced graph mining algorithms in conjunction with these runtime analysis tools, it may be possible to identify frequently-executed/time-consuming sections of code, and map them onto available accelerator hardware.

Another possible area of future research would be the creation of parameterized libraries for accelerating common computation-intensive kernels for different types of accelerators. For instance, Asanovic et al [40] have identified what they call the "13 Dwarfs" – 13 distinct kernels that they believe represent the most computation intensive sections of current and future applications. These 'dwarfs' will provide the greatest challenge for future parallel processing systems.

Utilizing our work in logic minimization through enumeration, we can explore the most efficient implementations/mappings of these kernels onto various FPGA architectures. We can then attempt to build a parameterized library of these computational kernels. This library of kernels would be pre-compiled to incorporate support for multiple types of accelerators through application of our efficient graph mining algorithms on the kernel's CFG/DFGs. Utilizing our library of kernels, an application programmer would then be able to invoke the appropriate kernel he requires, provide the relevant parameters, specify the accelerator and other characteristics of his target system, and then compile his application. The process of efficiently implementing the computationally intensive kernel onto the accelerator would thus be fully automated.

References

[1] J. Yang, S. A. Savari, and O. Mencer, "An Approach to Graph and Netlist Compression," to appear in Proceedings of the 2008 IEEE Data Compression Conference (DCC'08), Snowbird, UT, March 2008.

[2] N. Clark, H. Zhong, and S. Mahlke, "Processor Acceleration through Automated Instruction-set Customization," in MICRO, San Diego, CA, Dec. 2003.

[3] F. Sun, S. Ravi, A. Raghunathan, and N. Jha, "A Scalable Application-specific Processor Synthesis Methodology," in ICCAD, pages 283–290, San Jose, CA, Nov. 2003.

[4] K. Atasu, G. Dündar, and C. Ozturan, "An Integer Linear Programming Approach for Identifying Instruction-set Extensions," in CODES+ISSS 2005, Jersey City, NJ, Sept. 2005.

[5] R. Leupers et al, "A Design Fow for Configurable Embedded Processors based on Optimized Instruction-set Extension Synthesis," in Design Automation and Test in Europe, Munich, Germany, Mar. 2006.

[6] K. Atasu, L. Pozzi, and P. Ienne, "Automatic Application-specific Instruction-set Extensions Under Microarchitectural Constraints," in 40th DAC, Anaheim, CA, June 2003.

[7] P. Bonzini and L. Pozzi, "Polynomial-time Subgraph Enumeration for Automated Instruction-set Extension," in Design Automation and Test in Europe, pages 1331–1336, Nice, France, Apr. 2007.

[8] J. Cong, Y. Fan, G. Han, and Z. Zhang, "Application-specific Instruction Generation for Configurable Processor Architectures," in FPGA 2004, Monterey, CA, Feb. 2004.

[9] P. Yu and T. Mitra, "Scalable Custom Instructions Identification for Instruction-set Extensible Processors," in CASES 2004, Washington, DC, Sept. 2004.

[10] K. Atasu et al, "Optimizing Instruction-set Extensible Processors Under Data Bandwidth Constraints," in DATE, pages 588–593, Nice, France, Apr. 2007.

[11] K. Atasu et al, "Fast Custom Instruction Identification by Convex Subgraph Enumeration," Proceeding of the 19th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP), Leuven, Belgium. July, 2008.

[12] N. Pothineni, A. Kumar, and K. Paul, "Application Specific Datapath with Distributed I/O Functional Units," in VLSI Design, pages 551–558, Hyderabad, India, Jan. 2007.

[13] A. K. Verma, P. Brisk, and P. Ienne, "Rethinking Custom ISE Identification: A New Processor Gnostic Method," in CASES, pages 125–134, Salzburg, Austria, Sept. 2007.

[14] R. M. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP Completeness," W. H. Freeman and Co., New York, 1979.

[15] A. Sangiovanni-Vincentelli, A. El Gamal, and J. Rose, "Synthesis Methods for Field Programmable Gate Arrays," Proceedings of IEEE, pp. 1057–1083, July 1993.

[16] V.N. Kravets and P. Kudva, "Implicit Enumeration of Structural Changes in Circuit Optimization," Proc. DAC, pp. 438–441, 2004.

[17] J. Nievergelt, "Exhaustive Search, Combinatorial Optimization and Enumeration: Exploring the Potential of Raw Computing Power," Proc. Conf. on Current Trends in Theory and Practice of Informatics, LNCS 1963, pp. 18–35, 2000.

[18] J. Yang, S. A. Savari, and O. Mencer, "Lossless Compression Using Two-Level and Multilevel Boolean Minimization," Proceedings of the IEEE 2006 Workshop on Signal Processing Systems (SiPS'06), Banff, AB, Canada, October 2-4, 2006.

[19] J. Yang, S. A. Savari, and O. Mencer, "An Approach to Graph and Netlist Compression," to appear in Proceedings of the 2008 IEEE Data Compression Conference (DCC'08), Snowbird, UT, March 2008.

[20] T. M. Cover and J. A. Thomas, "Elements of Information Theory," 2nd Edition, Wiley, New Jersey 2006.

[21] S. Savari and C. Young, "Comparing and Combining Profiles," Proceedings of the 2nd Workshop on Feedback-Directed Optimization, Haifa, Israel, November 1999, Best Presentation Prize.

[22] O. Mencer, "PAM-Blox II: Design and Evaluation of C++ Module Generation for Computing with FPGAs," International Symposium on Field Programmable Custom Computing Machines (FCCM), Napa Valley, April 2002.

[23] O. Mencer, M. Platzner, M. J. Flynn, M. Morf, "Object-oriented Application Specific Compilers for Programming FPGAs," IEEE Trans. VLSI, Special Issue on Reconfigurable Computing, Feb. 2001.

[24] T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer, W. Luk and P. Y. K. Cheung, "Reconfigurable Computing: Architectures and Design Methods," IEE Proceedings on Computers and Digital Techniques, Vol. 152, no. 2, pp. 193–207, 2005.

[25] D. Lee, A. A. Gaffar, O. Mencer, W. Luk, "Optimizing Hardware Function Evaluation," IEEE Transactions on Computers, Dec. 2005.

[26] Oskar Mencer, Luc Semeria, Martin Morf, and Jean-Marc Delosme, "Application of

Reconfigurable CORDIC Architectures," Kluwer Journal of VLSI Signal Processing, Special Issue, Reconfigurable Computing, March 2000.

[27] K. Atasu, R. Dimond, O. Mencer, and W. Luk, "Towards Optimal Custom Instruction Processors," IEEE Hot Chips Conference, Stanford, August 2006. ¨

[28] K. Atasu, R. Dimond, O. Mencer, W. Luk, C. Ozturan, and G. Dündar, "Optimizing Instruction-set Extensible Processors Under Data Bandwidth Constraints," Design Automation and Test in Europe Conference and Exhibition, Nice, France, April 2007

[29] R. Dimond, O. Mencer, W. Luk, "Automating processor customisation: optimized memory access and resource sharing," Design Automation and Test in Europe, March 2006.

[30] Oskar Mencer, Marco Platzner, "Dynamic Circuit Generation for Boolean Satisfiability in an Object-oriented Design Environment", HICCS (ConfigWare Track), Jan. 1999.

[31] Oskar Mencer, "ASC: A Stream Compiler for Computing with FPGAs," IEEE Trans. CAD, Aug. 2006.

[32] Katherine Compton, Scott Hauck, "Reconfigurable Computing: A Survey of Systems and Software," ACM Comput. Surv. 34(2): 171-210 (2002)

[33] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krger, Aaron E. Lefohn, and Timothy J. Purcell, "A Survey of General-purpose Computation on Graphics Hardware," Computer Graphics Forum, 26, 2007

[34] J. Henkel, "Closing the SoC Design Gap," IEEE Computer, vol. 36, no. 9, pp. 119-121, Sept. 2003.

[35] D. A. Patterson and J. L. Hennessy, "Computer Architecture: A Quantitative Approach", Morgan Kaufmann Publishers, 1996.

[36] http://comparch.doc.ic.ac.uk/index.php?pageid=20

[37] www.nallatech.com

[38] www.drc.com

[39] www.spec.org

[40] K. Asanovic, R. Bodik, B. Catanzaro, et al, "The Landscape of Parallel Computing Research: A View from Berkeley," Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, December 2006.

[41] Wilson, R., French, R., Wilson, C., Amarasinghe, S., Anderson, J., Tjiang, S., Liao, S.-W., Tseng, C.-W., Hall, M., Lam, M., and Hennessy, J., "SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers. ACM SIGPLAN Not. 29, 12, Dec. 1994.

[42] V.N. Kravets and P. Kudva, "Implicit Enumeration of Structural Changes in Circuit Optimization," Proc. DAC, pp. 438–441, 2004.

[43] J. Nievergelt, "Exhaustive Search, Combinatorial Optimization and Enumeration: Exploring the Potential of Raw Computing Power," Proc. Conf. on Current Trends in Theory and Practice of Informatics, LNCS 1963, pp. 18–35, 2000.

[44] Ronald L Graham, Donald E Knuth, Oren Patashnik, "Concrete Mathematics: A Foundation for Computer Science," Addison-Wesley, 1989.

[45] Donald E Knuth, "A Draft of Section 7.2.1.3: Generating All Combinations," The Art of Computer Programming: Pre-Fascicle 3A, 2005.

[46] A. J. Elbirt, C. Paar, "An FPGA Implementation and Performance Evaluation of the Serpent

Block Cipher,", ACM/SIGDA International Symposium on FPGAs, pp. 33-40, 2000.

[47] J. R. Hauser, J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor," IEEE Symposium on Field-Programmable Custom Computing Machines, 1997.

[48] H. J. Kim, W. H. Mangione-Smith, "Factoring Large Numbers with Programmable Hardware", ACM/SIGDA International Symposium on FPGAs, pp. 41-48, 2000.

[49] Luc Buatois and Guillaume Caumon and Bruno Lévy, "Concurrent number cruncher - A GPU implementation of a general sparse linear solver," International Journal of Parallel, Emergent and Distributed Systems, to appear 2008.

[50] Ivan S. Ufimtsev and Todd J. Martínez, "Quantum Chemistry on Graphical Processing Units - Strategies for Two-Electron Integral Evaluation", Journal of Chemical Theory and Computation, 4(2), pgs: 222-231, 2008.

[51] http://www.gpgpu.org

[52] http://www.openfpga.org