Appendix A: A functional programming approach to hardware acceleration of algorithms

Mary Sheeran (ms@chalmers.se, +46 31 772 1013)

CSE Dept., Chalmers

Purpose and aims

The aim of this proposal is to develop methods and tools to enable large scale acceleration of algorithms using reconfigurable hardware (Field Programmable Gate Arrays, FPGAs). FPGAs currently contain resources other than just a fabric of computing elements; examples include fast carry chains, embedded DSP circuits that run much faster than the reconfigurable fabric, and embedded processors. These additional resources make FPGAs into very powerful computing platforms, but also demand sophisticated methods if they are to be efficiently exploited. Previous knowledge from the design of algorithmic blocks (such as fast binary adders) for implementation on full custom hardware is not simply transferrable to these augmented FPGAs. A new approach is needed, and we propose one based on functional programming and search.

The application areas that we aim to support demand fast data-paths, as distinct from more control oriented computations. They are medium scale cryptography (which demands Montgomery multiplication and exponentiation), large scale correlators and convolvers for astronomy and, finally, fully homomorphic encryption (FHE), which (currently) demands arithmetic on millions of bits, and which, if successfully made practical, will enable secure cloud computing. The proposed research is distinguished from current approaches by:

- the strong emphasis on data-paths. This leads to the need to have fine control over layout on the FPGA and also makes starting from a sequential C-like description infeasible.
- a programming language based approach to exploiting additional resources (including processors) on the FPGA, combined with the use of search
- the sheer scale of the circuits needed in fully homomorphic encryption
- initial strong results on parallel prefix networks (a key building block in fast adders)

Survey of the Field

FPGA programming

FPGA programming is steadily moving towards greater use of High Level Synthesis, but this is hard to reconcile with having fine control over resource use, particularly for data-path implementation. At ENS Lyon, the FloPoCo project aims to develop high performance floating point cores for FPGAs, and it has found the need to develop new methods to implement even binary adders on FPGAs, particularly with pipelining is required [5]. A number of authors have considered ways to map chaining computations onto the fast carry chains of FPGAs, enabling implemenation of many more operations that just ripple carry adders (e.g. [7, 14]). Our aims are similar, but we want to put control of the mapping into the hands of the programmer (assisted by the use of search). Researchers currently implementing very large and regular digital signal processing arrays for use in astronomy have found that current design tools make the necessary control of layout on the FGPA both slow and painful. The development of tool-chains for FPGA programming has concentrated on relatively small scale control-oriented applications, and on harnessing *software* developers – with the result that there is much work on C-like languages for FPGA design. This approach is inappropriate for our very high performance, very highly parallel applications.

Domain Specific Languages (DSLs)

Domain specific languages (DSLs) can be used to give fine control of resources, with the restriction to a specific domain being what makes this feasible. Examples in hardware (FPGA) design include our work with Singh and later Claessen on Lava, a DSL for hardware netlist generation [3]. Lava has been influential and there are now several new implementations (for example [10], which provides new abstractions, including ways to describe and control memories). We see strong possibilities for collaboration here. The ability to control geometry is a key ingredient of the success of Lava in real applications (as implemented by Singh at Xilinx). Working with Intel, we explored the control of geometry at an even finer level of detail in work on Wired, which enables wire-aware low level hardware design [2]. This proposal aims to return to work on Lava, and on sophisticated circuit generation methods, and to extend it to take account of developments in FPGAs that now combine the FPGA fabric with other computing resources.

Cost models and dynamic programming

The use of integer linear or dynamic programming has a strong history in arithmetic circuit generation (e.g. the classic work on optimal multipliers [21]). Such methods are closely associated with cost models. In our own approaches to circuit generation, we have relied heavily on Non-Standard Interpretation, a variant on classic abstract interpretation, to estimate costs for the purposes of controlling the search for sufficiently good solutions. We will need to use much more sophisticated cost modelling if we are to succeed in exploiting FPGAs plus extra resources such as embedded DSPs or multipliers. We believe that we will be able to build on work by Blelloch and his co-workers on cost models for parallel functional programming [20]. Although our context is rather different, we see strong parallels between our envisaged cost models and those proposed by Blelloch et al. Both lines of research place a strong emphasis on a programming language (rather than a machine model) based approach to algorithm discovery, mapping and profiling.

Cryptography, our main application area

Modular multiplication is an important building block in modern cryptographic algorithms. Chow et al have recently studied the implementation on FPGA of a recursive Karatsuba-based Montgomery multiplier [4]. We will start with this recursive construction as one of our case studies as it seems well suited to the use of search-based generation techniques and will force us to adapt those methods to real FPGA generation (see below).

Our main motivating application is, however, fully homomorphic encryption (FHE), which was shown possible (if very difficult) in Gentry's thesis from 2009 [8]. Gentry's result has created an enormous splash. In the US, DARPA has appointed Galois, Inc. as research integrator for the $20m PROCEED program (Programming Computation on Encrypted Data) whose goal is to make it feasible to execute programs on encrypted data without having to decrypt the data first. "If we are successful with PROCEED, it fundamentally changes the calculus for computations in untrusted environments on computer systems of unknown provenance. The potential implications for the cybersecurity of cloud computing architectures are profound" states DARPA Director Regina Dugan in testimony submitted to the US House Subcommittee on Emerging Threats and Capabilities, March 1, 2011. Homomorphic encryption offers the possibility of being able to delegate the *processing* of data, without having to give away *access* to it. Gentry has provided an encryption scheme that "keeps data private, but that allows a worker that does not have the secret decryption key to compute any (still encrypted) result of the data, even when the function of the data is very complex" and that thus "helps make cloud computing compatible with privacy" [9]. The downside is that all known FHE schemes are computationally extremely expensive, as they encode the decryption function (inefficiently) as a circuit, and then, in the *Evaluate* algorithm, replace each bit in that circuit with a large ciphertext that encodes that bit. Much work by many researchers will be needed to make a truly practical FHE scheme. Finding ways to implement and run the very large boolean circuits that result is what interests us, partly because it provides the strictest of tests of our proposed FPGA design methods, and partly because the need to implement very large circuits opens new research questions about key algorithms.

We have strong links to Galois Inc., which works very much with functional programming and domain specific languages. Part of Galois Inc's work on the PROCEED project will use their Cryptol DSL for the development and verification of cryptographic algorithms [13] as a means to demonstrate research results in the project. The generation of VHDL (for FPGA programming) from Cryptol was inspired by the applicant's DPhil thesis and by her early work on retiming [16]. Andy Gill, who did much of this work at Galois, is now back in academia and we are planning collaboration, as he continues to work with his new version of Lava, and with FPGAs for algorithmic computations [10]. While we appreciate the expressiveness that Cryptol's advanced type system brings, we are firmly convinced of the need for an expressive meta-programming layer if we are to meet the enormous challenges of fully homomorphic encryption, which currently involves (in one of its "almost homomorphic" parts) binary arithmetic on 13 million bit numbers.

Preliminary Results

DSLs for hardware design

Lava is a system that supports the design and verification of circuits [3]. It is an extensible domain specific language embedded in the standard functional programming language Haskell. Lava descriptions encode standard ways to build circuits (*connection patterns*) as higher order functions. The standard Haskell function `map` corresponds to placing a component on each element of a list of inputs (a bus). Lava includes a way to capture sharing in circuit descriptions, so that one can write what look like plain Haskell descriptions, and do not need to resort to heavier weight constructions (such as monads) when descrbing cyclic circuits.

For non-cyclic circuits, it is easy to describe circuits directly in Haskell, and to define various interpretations of them. For example, an important pattern is parallel prefix or scan. Given inputs $[x_0, x_1 \ldots x_{n-1}]$, the prefix problem is to compute each $x_0 \circ x_1 \circ \ldots \circ x_j$ for $0 \leq j < n$, for $\circ$ an associative, but not necessarily commutative, operator. In a construction attributed to Sklansky, one can perform the prefix calculation by first, recursively, performing the prefix calculation on each half of the input, and then combining (via the operator) the last output of the first of these recursive calls with each of the outputs of the second, see Figure 1. The Haskell description is parameterised on a fan structure, which both passes through its first input and applies the binary operator to that input and each of the remaining elements of its input list, to give an output list whose length is the same as that of the input list:

```
mkFan op (i:is) = i:[op i k | k <- is]

pplus = mkFan (+)
```

For example, `pplus [1..8]` gives the list `[1,3,4,5,6,7,8,9]`. The Haskell description of the algorithm contains two recursive calls of `skl`, each operating on roughly half of the input. The fan structure is used to combine the last output of the first of these with each of the outputs of the other:

```
skl _ [a] = [a]
skl f as  = init los ++ ros'
  where
    (los,ros) = (skl f las, skl f ras)
    ros'      = f (last los : ros)
    (las,ras) = splitAt (chalf (length as)) as

chalf n = n - n `div` 2 -- Ceiling of n/2
```

Now, `skl pplus [1..4]` is `[1,3,6,10]` and `skl pplus [5..8]` is `[5,11,18,26]`. The final fan structure applies to `(10 : [5,11,18,26])`, so that the result of `skl pplus [1..8]` is `[1,3,6,10,15,21,28,36]` Figure 2 shows another standard prefix network construction due to Ladner and Fischer [12]. Both of these diagrams were generated by *running* their Haskell descriptions using a suitable building block (parameter) that gathers the necessary information. It is also possible to run circuit descriptions in order to get cost estimations, and this can be done either after, or, more interestingly, during circuit generation.

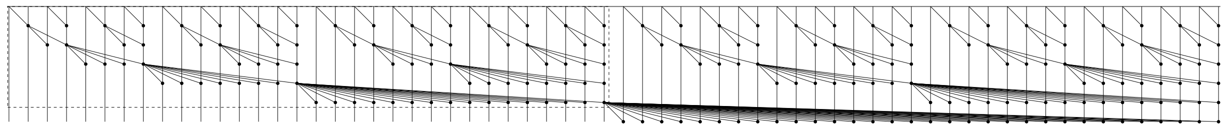

Fig. 1. The Sklansky construction for 64 inputs, illustrated using a diagrammatic notation for prefix networks. It recursively computes the parallel prefix for each half of the inputs; the dotted box shows the first of these recursive calls. It then combines the last output of that call with each of the outputs of the other recursive call.

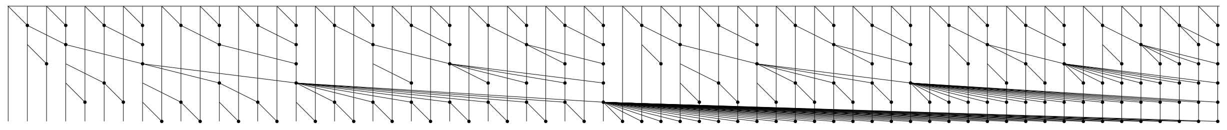

Fig. 2. The Ladner Fischer (LF) construction for 64 inputs. Note how the bottom of the left part of the network makes use of the entire network depth, unlike in the Sklansky construction.

More advanced generation methods

In Lava, we have explored the notion of *clever circuits* – circuits that have additional Haskell-level *shadow* parameters carrying non-functional properties, allowing them to adapt to their contexts *during circuit generation* [17]. We demonstrated the method on multiplier reduction trees [18]. In the multiplier reduction trees, the cells can be thought of as being placed initially, and it is only the wiring between them that is chosen during generation. In that work, the context consisted only of the shadow values capturing input delays. One can go further and have the context capture both input and required output delays. One can then enumerate and choose between a large number of possibilities for the entire topology of the network, using its recursive decomposition and dynamic programming.

We have had considerable success in doing this for parallel prefix networks. By choosing a good recursive decomposition, one can find good (that is small) networks, of fixed size but for large number of inputs, using a variety of different measure functions, and including constraints on fanout [19]. The resulting generator is pleasingly small and the results improve on known best solutions for depth size optimal networks. A key point here is that one is making significant use of the host language in writing sophisticated generators; so the fact that the DSLs we study are *embedded* is important.

This work has in turn led to the development of a new parallel prefix algorithm that does not require search, but that grew out of the insights gained from seeing the results of search in many contexts. In the new algorithm, the number of operators (the *size*) for a minimum depth network with $w = 2^n$ inputs approaches $3.5w$, while the Ladner Fischer algorithm approaches $4w$. This is a substantial improvement, different from and also improving on the more advanced of Fich's prefix constructions [6]; for instance, it requires 14662683 operators for $2^{22} = 4194304$ inputs, while the corresponding sizes for Ladner Fischer and Fich are 16580799 and 14851947 respectively. Upon the recent appearance of this result in the Journal of Functional Programming, we were contacted by a Russian complexity theorist, who had proven an *exact* lower bound for 2^n-input, depth n parallel prefix networks [15].

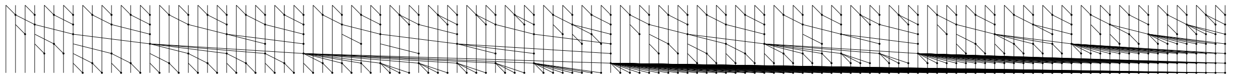

Fig. 3. The new construction for 128 inputs, depth 7. It uses 364 operators, compared to 369 for LF [12] (and 448 for Sklansky).For 256 inputs, the three sizes are 773 for our construction, 792 for LF and 1024 for Sklansky.

Sergeev has proved that the number of operators in a prefix network with 2^n inputs and depth n is bounded below by $(3.5 * 2^n) - (8.5 + 3.5 * (n\%2)) * 2^{(n/2)} + n + 5$, where % is the modulus function. The result relies on a sophisticated argument about the number of redundant operators that must appear in a network of shape similar to the Ladner Fischer construction. (Sergeev's result has so far only been published in preliminary form, and only in Russian. A slightly longer version will appear mid-year and will be translated to English.) The interesting thing is that our construction matches this bound *exactly*, and so is, to the best of our knowledge, the first presentation of an optimal (that is smallest possible) minimal depth prefix network construction. We feel that our success in solving this open problem in prefix network design was due to the use of functional programming plus search, and to the ease of experimentation that ensued. Our aim, now, is to develop this approach further to enable the implementation both of very large prefix networks and of other key algorithms such as sorting and median finding. This will involve both further work on the algorithms themselves and on ways to implement these algorithms on FPGAs.

Our concentration on prefix networks grew out of contacts at Intel Strategic CAD Labs, and has its basis in the fact that prefix networks are key elements both in arithmetic ciruits (where they provide a means to implement fast carry propagation in adders) and more generally in microprocessors, where, for example, they are used to implement priority encoders. There is much work on parallel prefix networks for VLSI circuits, but this does not transfer easily to FPGAs because of the additional resources, particularly fast carry chains. This is well explained in reference [1], which points out that there has been no systematic study of parallel prefix networks on FPGAs, and considers the problem of implementing larger prefix networks (up to 256 inputs) with word level rather than bit level operators. However, one of the chosen implementations is labelled Ladner Fischer, but is actually the Sklansky construction (a common misconception in papers and even text books). We speculate that the cause of this widespread misconception is a lack of suitably expressive programming tools to describe slightly irregular algorithms, and of associated tools to map those algorithms to the FPGA fabric, including the specific extra computing resources that the FPGA provides. At 256 inputs, implementing Sklansky rather than Ladner Fischer means using 1024 instead of 792 operators, a difference that would probably affect power consumption (which tends to depend on circuit size). One of our aims in this project is conduct a serious study of parallel prefix network implementation on FPGAs, first for medium scale (up to say 1024 inputs), and later for the gigantic prefix networks that will be needed for arithmetic on very large numbers, as required in fully homomorphic encryption.

Project plan

Planned tasks in the first phase of the project

Circuit needs of current cryptography Study the need for circuits in current crypto, with emphasis on Montgomery multiplication and exponentiation. Use these as guiding case studies (along with prefix networks) in the following tasks.

Making search more systematic Continue work on search in algorithm development. Make the approach more systematic. Develop combinators for search. Extend the notion of context to include richer constraints on the solution (e.g. the existence of carry chains that should be used). This will entail developing much more sophisticated cost models than those used during our work on circuit geneation so far.

First practical steps in exploiting and controlling additional computing resources: fast carry chains Working with real FPGA implementations, develop the above-mentioned search methods for exactly the case of fast carry-chains. With Satnam Singh, perform a comparative study of parallel prefix network implementation on FPGA.

Result at the end phase one (18 months) Serious study of medium scale prefix networks and Montgomery multipliers on FPGAs, with strong emphasis on programmer control of resources. This will include higher radix prefix networks, where results on optimality are lacking, so that work on the algorithms themselves is needed. This will form the basis for later work on implementing very large scale prefix networks.

Planned tasks in the second phase of the project

Clever circuits revisited Consider the case where the most natural way to express an algorithm is to make modifications (using the clever circuits idea) to one that does more but is very regular. An example is our earlier work on generating median networks from sorting networks [17], where it was possible to reduce the number of comparators needed to produce the median of 25 inputs (a common operation in graphics) from 99 to 96. This kind of application of clever circuits has not been much explored and is promising.

Further work on exploiting and controlling additional computing resources: DSPs and processors Extend work on ways to express use of computing resources + FPGA routing fabric. First consider DSPs (which provide fast arithmetic), and then processors. The need to make use of processors will place new demands on our algorithm decomposition methods, as one must now decide which sub-parts to implement in the processors. Our approach to this problem will be to develop programming oriented cost models, very much along the lines of those developed for parallel functional programming by Blelloch and his coworkers [20]. In the search-based prefix network generation described earlier, the context for the search is something akin to a hole that indicates position and delay on inputs and outputs, and into which the final network must fit. Now, the context is going to have to contain additional resources placed within the hole. The combinators for expressing the search will have to be more sophisticated, to give control over placement of functions on the FPGA fabric or on the additional computing resource.

The main area of application: fully homomorphic encryption Study fully homomorphic encryption and its need for very large arithmetic circuits. This application will force us to develop ways to decompose the algorithms and to play with space time trade-offs. For the extraordinary demands imposed by FHE, it may be necessary to consider new FPGA structures (along the lines of work by Hauck et al [11]).

Expected results at the end of the project

– Greater understanding of data-independent (circuit-like) parallel prefix algorithms both in theory and with regard to practical implementation at very large scale.
– Demonstration of implementations of very large scale arithmetic circuits (with parallel prefix as a key component) for use in FHE.
– Methods of programming combinations of FPGAs with other computing resources. The methods of programming FPGAs + CPUs will give first results in the search for hardware software codesign methods for algorithmic (rather than control oriented) problems.

Once we have the cost models and search combinators that enable easy programming of additional resources on the FPGA, success will be measured both in terms of performance and area of the generated applications, and of ease of programming new applications.

Collaboration

The Chalmers Functional Programming Group provides an excellent research environment for this proposal. The group is very well resourced. Our work in DSLs for software includes development of a DSL framework, which will be very useful in this project.

Singh, with whom we worked on Lava, is now at MSR, Cambridge. He has recently worked on methods for programming of heterogenous systems, and on ways to translate (ordinary) Haskell programs into circuits. He is visiting faculty, interacting strongly with our group. Both his recent research and his practical expertise in FPGA programming will benefit us. We plan to work together on FPGA implementations of cryptographic algorithms, and on the necessary cost models and search-based programming methods. Arvind, from MIT, is working on the use of the Bluespec DSL in hardware-software codesign. Our work is more data-path oriented, but we will doubtless also need some control-oriented aspects, just as the Bluespec work has incorporated circuit design patterns similar to ours. Harper's work with Blelloch on cost models has inspired us. It was John Launchbury, CTO of Galois Inc., who introduced the author to fully homomorphic encryption. The author has the privilege of meeting all of these researchers yearly at the IFIP working group on functional programming, of which she is a member. We expect during this project to establish research collaboration with Andy Gill (U. Kansas). We also plan collaboration with Marc Pouzet and Jean Vuillemin, who, with A. Cohen, are forming a group to work in synchronous programming at ENS, Paris. We have strong shared interests in algorithms, hardware design and functional programming languages. Mutual visits are planned. Thus, the proposed research will be supported by a strong network of colleagues and collaborators.

Budget and part of project plan, Overlap with existing projects

Sheeran is co-applicant on Hughes' VR frame grant "Putting functional programming to work: Software Design and Verification using Domain Specific Languages". In that proposal, we excluded work on hardware design, in order to keep the proposal focussed. We made the same decision in our recent (granted) proposal to SSF on resource aware functional programming. That proposal also concentrates entirely on software development methods. This proposal is about hardware design methods, and is designed to support the author's work in this area. The proposed research does not overlap with either of the above-mentioned proposals. This proposal aims to fund 100% of the proposed work.

The intention is to hire an assistant professor (forskarassistent) on the project, possibly from among our existing postdocs. Sheeran plans to spend approximately 25% of her time on this work (funded from other sources). The requested funding for equipment includes 20.000 SEK for FPGA boards.

Significance

Programming FPGAs with embedded carry chains is non-trivial and tends to be done on a case by case basis, with specialised module generators. We want to use the *programming language* to make it more possible for the user to control the final result, by building upon a library of search combinators.

The problem of how to program FPGAs with embedded computational resources such as processors is harder. Solving it will enable greater use of such platforms. The problem also bears some relationship to the more general question of how the transition from pure software to hardware-accelerated software should be made, or more generally how to do hardware software co-design. We don't want to attack the whole of that complex and so far alarmingly difficult problem, but to specialise our work to large data-paths, in order to make progress on investigating *simple* approaches. Any success here will have broad impact. The development of usable programming language based methods for expressing ways in which algorithms should be divided between CPUs and hardware accelerators would benefit applications such as baseband signal processing, where accelerators for key functions like Fast Fourier Transform are routinely used. Our methods could also ease the use of FPGA simulation of VLSI circuits in hardware verification (so-called emulation).

We expect to make progress in theoretical questions relating to key algorithms (such as parallel prefix or multipliers) – developing a programming language rather than a machine model oriented approach to this work on complexity results for real circuits. Parallel prefix is a central algorithm in computer science, as it sheds light on parallel implementations of an apparently sequential algorithm. It is also a key building block in data parallel software, such as that used in graphics processing. Results here will influence not only hardware but also software implementations.

The holy grail, though, is to find principled ways to design and implement the gigantic circuits that are currently known to be needed to achieve a practical form of fully homomorphic encryption. We have contacts with Galois Inc., who are the research integrator in the DARPA PROCEED initiative in this area, so there will certainly be a route to real application of our results, should they merit such application. The author has in recent years prioritised work on software development methods, and has successfully engaged with Swedish industry in that area. However, the combination of hardware design and functional programming continues to fascinate, and this new holy grail provides challenging research questions, highly qualified collaborators and a route to high impact.

References

1. N. Abbas, S. Derrien, P. Quinton, and S. Rajopadhye. Accelerating HMMER on FPGA using Parallel Prefixes and Reductions. In *Proc. IEEE Int. Conf. on Field-Programmable Technology (FPT'10)*. IEEE, 2010.
2. Emil Axelsson. *Functional Programming Enabling Flexible Hardware Design at Low Levels of Abstraction.* PhD thesis, CSE Dept., Chalmers University of Technology, 2008.
3. P. Bjesse, K. Claessen, M. Sheeran, and S. Singh. Lava: Hardware design in Haskell. In *International Conference on Functional Programming*. ACM Press, 1998.
4. Gary C.T. Chow, Ken Eguro, Wayne Luk, and Philip Leong. A Karatsuba-Based Montgomery Multiplier. *Int. Conf. on Field Programmable Logic and Applications*, pages 434–437, 2010.
5. F. de Dinechin, H.D. Nguyen, and B. Pasca. Pipelined FPGA Adders. In *Int. Conf. on Field Programmable Logic and Applications*. IEEE Computer Society, 2010.
6. Faith Ellen Fich. *Two problems in concrete complexity: cycle detection and parallel prefix computation.* PhD thesis, University of California, Berkeley, 1982.
7. Michael T. Frederick and Arun K. Somani. Beyond the Arithmetic Constraint: Depth-Optimal Mapping of Logic Chains in LUT-based FPGAs. In *Symposium on Field Programmable Gate Arrays*. ACM, 2008.
8. Craig Gentry. *A fully homomorphic encryption scheme.* PhD thesis, Stanford University, 2009.
9. Craig Gentry. Computing Arbitrary Functions of Encrypted Data. *Communications of the ACM*, March 2010.
10. A. Gill, T. Bull, A. Farmer, G. Kimmell, and E. Komp. Types and type families for hardware simulation and synthesis: The internals and externals of kansas lava. In *Trends in Functional Programming*, 2010.
11. S. Hauck, T.W. Fry, and M.M. Hosler. High-performance carry chains for FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(2), 2000.
12. Richard E. Ladner and Michael J. Fischer. Parallel prefix computation. *J. ACM*, 27(4), 1980.
13. J. R. Lewis and B. Martin. Cryptol: high assurance, retargetable crypto development and validation. In *Military Communications Conference, Volume 2*, pages 820–825. IEEE, 2003.
14. H. Parandeh-Afshar, P. Brisk, and P. Ienne. Exploiting fast carry-chains of FPGAs for designing compressor trees. In *Field-Programmable Logic and Applications*. IEEE, 2009.
15. Igor S. Sergeev. Some complexity estimations for parallel prefix schemes (in Russian). In *Proc. 10th Int. Seminar on Discrete Mathematics and its Applications*. Moscow State University, Feb. 2010.
16. M. Sheeran. Retiming and slowdown in Ruby. In G.J. Milne, editor, *The Fusion of Hardware Design and Verification*. North-Holland, 1988.
17. M. Sheeran. Finding regularity: describing and analysing circuits that are almost regular. In *Correct Hardware Design and Verification Methods*. LNCS 2860, Springer, 2003.
18. M. Sheeran. Generating fast multipliers using clever circuits. In *Formal Methods in Computer-Aided Design, FMCAD*, volume 3312 of *LNCS*. Springer, 2004.
19. M. Sheeran. Functional and dynamic programming in the design of parallel prefix networks. *Journal of Functional Programming*, 21(1), 2011.
20. Daniel Spoonhower, Guy E. Blelloch, Phillip B. Gibbons, and Robert Harper. Space Profiling for Parallel Functional Programs. *Journal of Functional Programming*, 20(5–6), 2011.
21. P. F. Stelling, C. U. Martel, V. G. Oklobdzija, and R. Ravi. Optimal Circuits for Parallel Multipliers. *IEEE Trans. Comp.*, 47(3), 1998.