Recursive Pipelined Genetic Propagation for Bilevel Optimisation

Shengjia Shao*, Liucheng Guo†, Ce Guo*, Thomas C.P. Chau*, David B. Thomas† and Wayne Luk*
*Department of Computing, †Department of Electrical and Electronic Engineering
Imperial College London
United Kingdom
E-mail: {shengjia.shao12, gl512, ce.guo10, cpc10, d.thomas1, w.luk}@imperial.ac.uk

Stephen Weston‡
‡Maxeler Technologies
London
United Kingdom
E-mail: weston@maxeler.com

***Abstract*—The bilevel optimisation problem (BLP) is a subclass of optimisation problems in which one of the constraints of an optimisation problem is another optimisation problem. BLP is widely used to model hierarchical decision making where the leader and the follower correspond to the upper level and lower level optimisation problem, respectively. In BLP, the optimal solutions to the lower level optimisation problem are the feasible solutions to the upper level problem, which makes it particularly difficult to solve. This paper proposes a novel hardware architecture known as Recursive Pipelined Genetic Propagation (RPGP), to solve BLP efficiently on FPGA. RPGP features a graph of genetic operation nodes which can be scaled to exploit hardware resources. In addition, the topology of the RPGP graph can be changed at run-time to escape from local optima. We evaluate the proposed architecture on an Altera Stratix-V FPGA, using a benchmark bilevel optimisation problem set. Our experimental results show that RPGP can achieve a significant speed-up against previous work.**

I. Introduction

The bilevel optimisation problem (BLP) is a two-level nested optimisation problem shown in (1). In BLP, the lower level optimisation problem (f) appears as a constraint of the upper level optimisation problem (F).

$$\begin{array}{lll} \max\limits_{x \in X} & F(x,y) & \\ \text{subject to} & G(x,y) \le 0 & \\ & \min\limits_{y \in Y} & f(x,y) \\ & \text{subject to} & g(x,y) \le 0 \end{array} \tag{1}$$

BLP is used to model two-level hierarchical decision making where two agents are involved: the *leader* and the *follower*. The leader corresponds to the upper level optimisation problem in BLP, has control over x, and optimises x with respect to the objective $F(x, y)$, constraints $G(x, y)$ and the follower. Meanwhile, in the lower level problem, the follower has control over y and optimises y with respect to the objective $f(x, y)$ and constraints $g(x, y)$. When the leader makes decision x, he will take account of the follower's potential response(s) - the optimal solution(s) to the lower level problem given x. The leader can foresee the follower's potential reactions. The leader has the initiative to choose an optimal x^* such that when the follower reacts as predicted, the overall result is the optimal for the leader. In the setting of BLP, the leader effectively guides the follower. There are many practical problems that fit BLP's structure very well [1]. A common case is the interaction between the government's policy and the industry or person affected [2], [3]. In [2], the leader is the government who sets a taxation policy with the aim of maximising tax revenue; the follower is a mining company who wants to maximise its profit under the limitation of the tax. In [3], toll gate setting and drivers' reaction are modelled as a BLP. In other areas such as corporation management [4] and engineering design [5], BLP is also found to be a powerful modelling approach.

While being useful, BLP is intrinsically very challenging to solve. This is because for each upper level candidate x_0, the lower level optimisation task must be solved to find the lower level optimum $y_0 = \underset{y \in Y}{\text{argmin}}\ f(x_0, y)$, which will be used to evaluate upper level objective function $F(x_0, y_0)$. Additionally, the two sets of constraints can often be inconsistent, which brings extra difficulty for convergence. Even in the linear case, BLP is known to be NP-hard [1]. As a result, heuristics, such as genetic algorithms, are the major ways to solve BLP. When using heuristics, solving a BLP often requires solving the lower level optimisation problem many times, which means the lower level objective function $f(x, y)$ will be evaluated a large number of times, such as 1 million [6]. To evaluate the lower level objective function a very large number of times is extremely time consuming for CPU, especially when these functions are non-trivial. As a result, objective function evaluations are the major performance overhead.

In this paper, we present a novel hardware-based genetic algorithm to solve BLP efficiently on FPGA. The entire system is built on FPGA so that objective function evaluations be achieved efficiently using a pipeline architecture. The proposed hardware structure, Recursive Pipelined Genetic Propagation (RPGP), extends recent work on Pipelined Genetic Propagation (PGP) to the bilevel case. It uses distributed memories to store candidates, thus avoiding the memory-access bottleneck of traditional FPGA-based genetic algorithms which use a single central memory. Also, the connections between genetic operation nodes can be changed at run-time to help escape from local optima. To the best of our knowledge, we are the first to solve BLP on FPGA. The main contributions of this paper are summarised as follows:

- A hardware-based BLP solver which operates solely on FPGA. The highly pipelined function evaluation modules in hardware greatly reduce function evaluation overhead that conventional CPU-based BLP solvers suffer from.
- The Recursive Pipelined Genetic Propagation (RPGP) architecture. RPGP is a novel genetic algorithm which employs distributed memories to avoid memory I/O bottleneck and dynamic topology switching to escape from local optima in both upper and lower level.
- Implementation of RPGP on an Altera Stratix-V FPGA and experimental evaluation using benchmark bilevel optimisation problems. The proposed RPGP system is found to be significantly faster than previous work.

The rest of this paper is organised as follows. Section II covers related background. Section III details our RPGP hardware structure. Section IV presents experimental evaluation. Section V discusses performance results. Finally, Section VI reaches to the conclusion and suggests future work.

II. Background

A. Solving BLP

As mentioned in Section I, the major computational overhead of solving BLP is a very large number of lower level objective function $f(x, y)$ evaluations. Therefore, theoretical research has been mainly focused on making $f(x, y)$ less computationally expensive to evaluate and reducing the number of function evaluations. In the simplest case, an analytical solution of the lower level problem is available, i.e. $y_0 = \underset{y \in Y}{\operatorname{argmin}}\, f(x_0, y) = y(x_0)$. In this case, BLP is reduced to a single level optimisation problem. Besides, if the lower level optimisation problem is differentiable, it is advisable to replace it by its Karush-Kuhn-Tucker conditions, which also reduces BLP into single level [7]. Another alternative is to replace lower level objective function by its quadratic approximation, which is a trade-off between accuracy and speed [8]. These methods usually assume the optimisation problem to have desired mathematical properties, such as smoothness and/or convexity. Unfortunately, this may not be the case for real world problems. In the general case, the common approach is nested optimisation, using heuristics to solve the lower level optimisation problem for each upper level candidate [9]. Nested heuristic optimisation approaches make no assumption on BLP's mathematical properties, but they are computationally expensive.

B. Genetic Algorithm on FPGA

As a customisable parallel computing platform, FPGA is a natural fit for genetic algorithms. The first FPGA-based genetic algorithm was proposed by Scott et al. in 1995 [10]. Since then, various designs and implementations have been proposed. An automatic platform for FPGA-based genetic algorithms is presented in [11], which enables the user to enter the optimisation problem and genetic algorithm parameters in a high level language and generates FPGA implementation automatically. However, these genetic algorithms usually use a large central memory to store the population and wait for all offsprings to be pooled into the population before starting next iteration of evolution. This will result in a memory I/O problem due to limited memory ports and pipeline stall due to the synchronisation in each iteration. A new hardware structure, Pipelined Genetic Propagation (PGP), is proposed in a recent paper [12]. In PGP, genetic operation nodes (selection, mutation and crossover) are connected in a graph and there is no central memory to store the entire population. Instead, each selection node has a small local on-chip memory to store a group of candidates. On every clock cycle, a new candidate comes into the selection node and an old candidate from local memory goes out. The better one between the newcomer and the old candidate will be written back to the selection node's local memory. In this manner, there is no memory I/O bottleneck. Also there is no need of global synchronisation, so PGP graph can operate in a fully-pipelined manner without stalling.

To the best of our knowledge, all previous FPGA-based systems are intended for single level optimisation with no support for BLP. In this paper, we present the Recursive Pipelined Genetic Propagation (RPGP) structure, which extends PGP to the bilevel case and is the first FPGA-based genetic algorithm for bilevel optimisation.

III. Hardware Design

We present the details of our RPGP structure in this section. RPGP is a nested optimisation approach using genetic algorithm, so it can be applied to a wide range of BLP problems regardless of smoothness or convexity. In RPGP, we use PGP to solve a lower level optimisation problem for each upper level candidate. Therefore, we put PGP graph for lower level optimisation into the selection node of upper level optimisation. Meanwhile, the upper level selection, mutation and crossover nodes form another PGP graph. Consequently, our RPGP system is essentially small PGP graphs nested within a large PGP graph, which is a 'recursive' version of PGP.

A. Lower Level PGP

The lower level PGP searches for the optimal lower level candidate for each upper level candidate, i.e. to find $y_0 = \underset{y \in Y}{\operatorname{argmin}}\, f(x_0, y)$ for each x_0. This can also be interpreted as finding the follower's optimal response given the leader's action. The lower level PGP is composed of 4 lower level selection nodes, 2 lower level mutation nodes, 2 lower level crossover nodes and a lower level controller. The number of selection, mutation and crossover nodes can be adjusted to

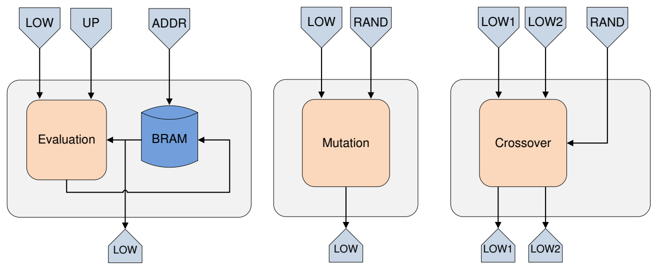

Fig. 1. Lower level PGP nodes: selection, mutation and crossover.

fully exploit FPGA resources. Figure 1 shows lower level selection, mutation and crossover nodes.

1) Lower Level Selection: A lower level selection node is composed of an input for lower level candidate, an input for upper level candidate, an input for BRAM address and an output for lower level candidate. On each cycle, a new lower level candidate comes in, and is evaluated and compared against a random selected candidate from existing population. The random candidate goes to the output and the better one of the two is written back to BRAM. As the evolution process goes on, the population in the BRAM become better and better. The BRAM address is generated on FPGA using an LUT-optimised uniform random number generator [13].

2) Lower Level Mutation: A lower level mutation node is composed of an input for lower level candidate, an input for random numbers, and an output for mutated lower level candidate. The random number is generated on FPGA using an LUT-optimised Gaussian random number generator [14]. The random numbers follow normal distribution $N(0, \sigma)$, where the variance σ can be specified at the beginning of computation. The benchmark problems used in this paper are numerical optimisation problems, so both the lower and upper level candidate are vectors of real numbers. In this case, mutation is carried out by adding Gaussian random numbers to candidate vectors. The mutation mechanism can be modified for different kinds of problems.

3) Lower Level Crossover: A lower level crossover node is composed of two lower level candidate inputs (‘parents’), two lower level candidate outputs (‘children’) and a random number input. For real-valued vectors, crossover is carried out by conditionally exchange elements between the parents, controlled by random numbers. The random number is also generated by the Gaussian random source [14]. The crossover mechanism can be modified for different kinds of problems.

4) Lower Level PGP graph: The lower level genetic operation nodes above are connected to the lower level PGP controller, which is essentially a complex crossbar controlling data paths. Figure 2 shows the structure of lower level PGP graph. The lower level PGP graph performs lower level optimisation for the input upper level candidate, and outputs corresponding optimal lower level candidate. To make use of FPGA’s flexibility, the special feature of PGP is that the crossbar configuration can be switched at run-time, which effectively offers another dimension of mutation. As shown

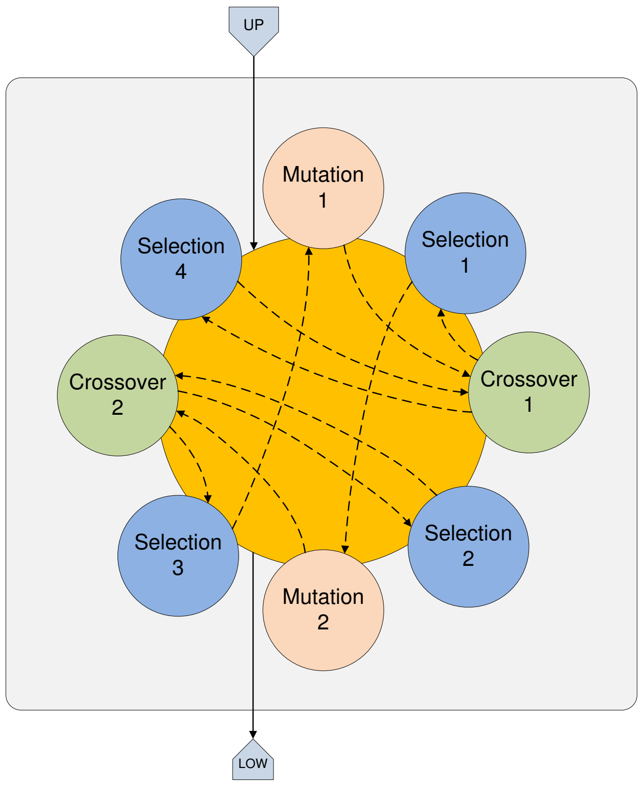

Fig. 2. Lower level PGP graph. It performs lower level optimisation for each upper level candidate and outputs corresponding optimal lower level candidate. The connections between selection, mutation and crossover nodes can be switched at run-time. The dynamic topology switch helps PGP to escape from local optima.

in the PGP paper, the dynamic topology can help the genetic algorithm escape from local optima [12].

B. Upper Level PGP

Upper level optimisation is also carried out by a PGP graph composed of several selection, mutation and crossover nodes. The upper level mutation and crossover nodes have the same structure as their lower level counterparts, so we will not elaborate them again. Note that although very similar, the upper/lower level mutation and crossover nodes are not interchangeable in general, because lower and upper level candidates may have different formats. On the other hand, the upper level selection node is much different from lower level selection. In fact, the lower level PGP graph is encapsulated inside the upper level selection node to form a recursive PGP.

1) Upper Level Selection: As required by nested optimisation, for each upper level candidate x_0, we need to find its corresponding optimal lower level candidate $y_0 = \operatorname*{argmin}_{y \in Y} f(x_0, y)$. Then the fitness of x_0 is evaluated by the upper level objective function $F(x_0, y_0)$. Note that lower level optimality is a strict requirement for solving BLP, because feeding a non-optimal y into $F(x_0, y)$ may yield a value better than the true value $F(x_0, y_0)$, which may lead the genetic algorithm to converge into a false upper level

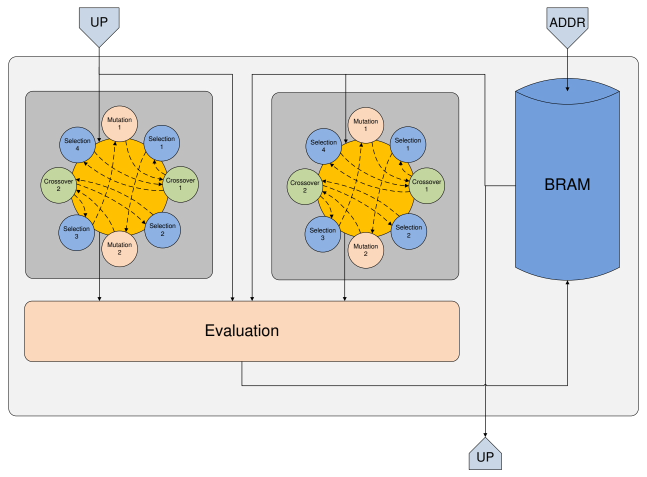

Fig. 3. Upper level selection node. The lower level PGP is nested inside to find the lower level optimum for the upper level candidates.

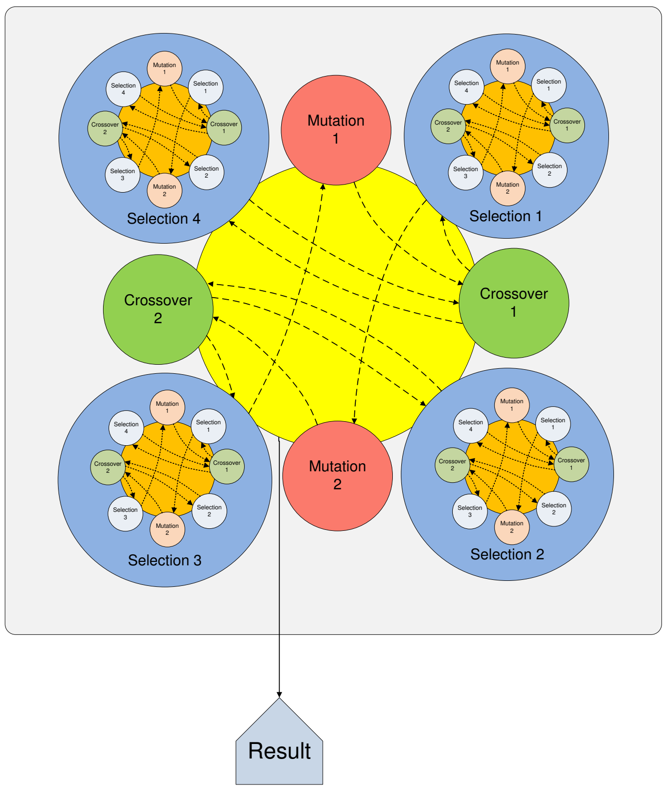

Fig. 4. Upper level PGP graph (Recursive PGP)

optimum. Intuitively, a non-optimal y with respect to x_0 means the follower is sacrificing himself, which is not the truth. For example, given a tax regulation x_0, a company will try to find an optimal solution y_0 to avoid tax as much as possible. So when planning taxation policy, the government must take y_0 into account, rather than a non-optimal y.

With this in mind, we put the lower level PGP graph inside the upper level selection node, so that optimal lower level candidate y_0 will go to the upper level evaluator $F(x, y)$ together with upper level input x_0. Figure 3 shows the structure of upper level selection node. The upper level candidate is fed into the lower level PGP graph in order to find its corresponding optimal lower level solution. Then the upper level and corresponding optimal lower level candidates are evaluated and compared with an existing candidate from BRAM. The existing candidate loaded form BRAM will be propagated to the output port, and the better of the two will be written back into BRAM. BRAM address is generated on FPGA using a uniform random number generator [13].

2) *Upper Level PGP graph (Recursive PGP):* The upper level PGP graph is composed of upper level selection, mutation and crossover nodes. Figure 4 shows the structure for upper level PGP. Note that lower level PGP is instantiated within upper level selection nodes, so we name the system as 'Recursive Pipelined Genetic Propagation (RPGP)'. The upper level controller in the center controls the data paths between upper level selection, mutation and crossover nodes. Similar to the lower level PGP controller, the data paths in the upper level PGP can be switched at run-time in order to escape from local optima in the upper level.

When compiling the RPGP system, the user will need to specify the lower level objective function $f(x, y)$, the upper level objective function $F(x, y)$, and constraints. When running the FPGA system, parameters for genetic operations are uploaded to RPGP at the beginning, such as the standard deviation (σ) for the Gaussian random number generator, lower and upper level thresholds to determine convergence. To minimise CPU-FPGA I/O overhead, random numbers are generated on FPGA, so RPGP does not require any inputs from CPU when running. RPGP will send final results back to CPU when the global optimum is found.

IV. Experimental Evaluation

A. System Specification

The number of selection, mutation and crossover nodes in both lower level and upper level are customisable to exploit the available resources on FPGA. The RPGP configuration used in this paper corresponds to Figure 4:

- Lower Level: Selection$\times$4, Mutation$\times$2, Crossover$\times$2
- Upper Level: Selection$\times$4, Mutation$\times$2, Crossover$\times$2

The proposed RPGP system is described using the MaxJ dataflow programming language by Maxeler Technologies. User will need to specify the BLP problem in a MaxJ file, then function evaluators $F(x, y)$ and $f(x, y)$ will be instantiated within RPGP. Currently there are two PGP topologies available for the lower level controller and the upper level controller. For the lower level PGP, the topology will be switched every 16384 clock cycles until optimal solution is found; for upper level PGP the topology will be switched every 16384*16384 cycles until the global optimum is found. The 16384 and 16384*16384 thresholds are empirical numbers which can be configured by the user.

B. Benchmark Problems

To evaluate the proposed RPGP system, we use four benchmark BLP problems proposed in [6]. These BLP problems have different properties:

- SMD1: Both the lower and upper level problems are convex. The two levels cooperate with each other.
- SMD2: Both the lower and upper level problems are convex. The two levels are in conflict.
- SMD5: The lower level has difficulties in convergence introduced by the Rosenbrock's function. Lower level optima lies in a narrow, parabolic valley. The two levels are in conflict.
- SMD6: The lower level contains infinitely many optimal solutions for each upper level candidate, but within the entire global solution set only one lower level point corresponds to upper level optimum. The two levels are in conflict.

These problems are scalable in terms of dimension. In our tests, we set the total dimension to 10: 5-dimensional lower level problem and 5-dimensional upper level problem.

C. Comparison

The proposed RPGP system is compared with BLEAQ, a recent CPU-based genetic BLP solver [15]. It makes quadratic approximations of the lower level problem whenever possible to boost performance. The BLEAQ solver targets the benchmark problem set above, and is publicly available (www.bilevel.org). We use BLEAQ's open sourced MATLAB version in our tests.

D. Test Environment

The proposed RPGP system is built on Maxeler's MAX4 platform with an Altera Stratix-V 5SGSD8 FPGA. FPGA frequency is set to 100MHz. The BLEAQ program is running in 64-bit MATLAB R2012b environment on a server with dual Intel Xeon E5-2640 CPU @ 2.5GHz and 64GB DDR3-1333 memory. The operating system used is CentOS 6.4.

E. FPGA Resource Usage

Table I shows the resource usage of Stratix-V 5SGSD8 FPGA. Although the RPGP configuration and parallelism are the same for all problems, resource usage varies significantly. This is because the majority of the FPGA resources are taken by the upper level and lower level objective function evaluators. For different problems these functions are different, resulting in various resource usage. For the SMD1 problem in the 10-dimensional case, both the upper level and lower level objective function contain two $tan()$ functions, which are expensive to evaluate on hardware. In contrast, SMD6 objective functions do not involve such expensive functions, so its resource usage is much less.

F. Performance Evaluation

We compare the elapsed time for RPGP and BLEAQ to achieve the same level of accuracy. In our tests we set the error threshold to 0.01, and the elapsed time is shown in Figure 5. As can be seen in the figure, RPGP demonstrates significant speed-up against BLEAQ. The best acceleration number is achieved for SMD2 test problem, in which RPGP running on FPGA is 1210 times faster than a Xeon-based BLEAQ solver in software. We think this is probably because: a) SMD2 has the largest number of function evaluations; b) objective functions in SMD2 contain $log()$, which is expensive for CPU to evaluate.

TABLE I
RESOURCE USAGE OF STRATIX-V 5SGSD8 FPGA

Problem #	Logic Usage	DSP Usage	BRAM Usage
SMD1	94.43%	100.00%	100.00%
SMD2	46.00%	82.53%	66.46%
SMD5	38.56%	78.45%	45.93%
SMD6	33.28%	47.48%	45.46%

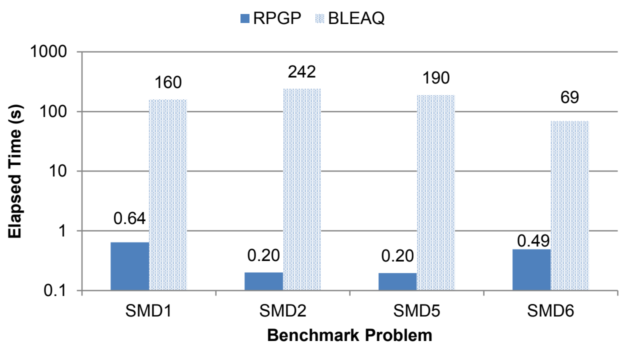

Fig. 5. Elapsed time of RPGP and BLEAQ to converge to $err < 0.01$. Speed-up factors achieved: 250(SMD1), 1210(SMD2), 950(SMD5), 141(SMD6).

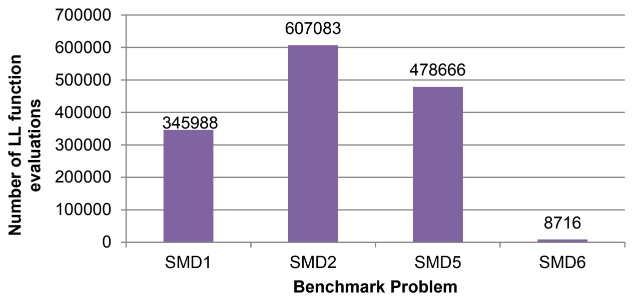

Fig. 6. Number of lower level objective function evaluations for BLEAQ

V. DISCUSSION

The major computational bottleneck for CPU-based BLP solvers is the huge number of lower level objective function evaluations. As can be seen from Figure 6, for test problems SMD1, SMD2 and SMD5, BLEAQ needs hundreds of thousands of LL evaluations, which is very time consuming for CPU, especially when the functions are non-trivial. For SMD6, the BLEAQ solver is lucky since it quickly converged to the threshold using about 10000 lower level function evaluations. Consequently, in Figure 5 the CPU time for

SMD6 is much shorter than those for other test problems. In general, the hardware speed-up is likely to be the most significant for the problems which are difficult to converge (need more evolution, so more function evaluations) and whose objective functions are costly to evaluate, such as those involving $tan()$ or $log()$. Apart from function evaluations, the genetic operations such as mutation and crossover are also expensive to run on a CPU, mainly because they need a significant amount of random numbers. For SMD6, although the number of function evaluations are low (such as 8716), it still takes 69 seconds to reach the 0.01 threshold.

On the other hand, in the FPGA-based RPGP system, all lower level and upper level function evaluators are built in hardware and highly-pipelined. Therefore, RPGP is actually indifferent to the time cost of evaluating objective functions, because on every cycle a fitness result will come out from the pipeline. As PGP is a non-generational approach, the lower level PGP graph do not need global synchronisation on each generation, therefore it is free from pipeline stall and has better efficiency.

In terms of area cost, RPGP is sensitive to the objective function's complexity. In each lower level selection node, there is a lower level objective function evaluator. As a result, as parallelism increases (more selection nodes), function evaluation units quickly occupy available FPGA resources. The RPGP systems for SMD1 and SMD6 have the same level of parallelism so they have the same number of objective function evaluation units. However, SMD1's $tan()$ functions use a lot more resources than SMD6's polynomials, resulting in an almost full FPGA. For random number generation, FPGA is very efficient. The proposed RPGP system has 4 LUT-optimised Gaussian number generators [14]. In total they only use about 5% logic and 4% BRAM.

VI. Conclusion and Future Work

Bilevel optimisation problem (BLP) is a nested optimisation problem in which one optimisation problem is nested within another as a constraint. BLP is a useful modelling approach for hierarchical decision making, where the leader and follower correspond to the upper level and lower level problem, respectively. However, BLP is an NP-hard problem and CPU-based heuristic solvers suffer from very large number of lower level objective function evaluation. In this paper, we propose the first FPGA-based BLP solver. Our system operates solely on FPGA and all function evaluators are built on hardware, which greatly reduces function evaluation overhead. The proposed Recursive Pipelined Genetic Propagation (RPGP) architecture employs distributed memories to avoid memory I/O bottleneck. Dynamic topology switching is used to escape from local optima in lower and upper level. Experimental results using four BLP benchmark problems show that the proposed RPGP system can achieve up to 1200 times speed-up compared to BLEAQ, a recent CPU-based BLP solver.

Future work involves improving RPGP's structure. As shown in Figure 3, there are two lower level PGP graphs in each upper level selection node. In RPGP's next version we will reduce this number to one, which will save a considerable amount of hardware resources. Also, we will do a thorough design space exploration, optimising number representation, FPGA frequency and parameters for genetic operation. Plus, we will include a more extensive performance evaluation in the future, using more benchmark problems and comparisons.

Acknowledgements

This work was supported in part by a Lee Family Scholarship, the China Scholarship Council, Altera, the Maxeler University Programme, the UK EPSRC grants EP/I012036/1 and EP/L00058X/1, the European Union Seventh Framework Programme under grant agreement number 318521, and the HiPEAC NoE.

References

[1] J. F. Bard, *Practical bilevel optimization: algorithms and applications*. Springer Science & Business Media, 1998, vol. 30.

[2] A. Sinha, P. Malo, A. Frantsev, and K. Deb, "Multi-objective stackelberg game between a regulating authority and a mining company: A case study in environmental economics," in *2013 IEEE Congress on Evolutionary Computation (CEC)*, pp. 478–485.

[3] L. Brotcorne, M. Labbé, P. Marcotte, and G. Savard, "A bilevel model for toll optimization on a multicommodity transportation network," *Transportation Science*, vol. 35, no. 4, pp. 345–358, 2001.

[4] J. F. Bard, *Coordination of a Multidivisional Firm Through Two Levels of Management*. Omega, 11(5), 1983.

[5] C. Kirjner-Neto, E. Polak, and A. Der Kiureghian, "An outer approximations approach to reliability-based optimal design of structures," *Journal of optimization theory and applications*, vol. 98, no. 1, pp. 1–16, 1998.

[6] A. Sinha, P. Malo, and K. Deb, "Unconstrained scalable test problems for single-objective bilevel optimization," in *2012 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8.

[7] J. Herskovits, A. Leontiev, G. Dias, and G. Santos, "Contact shape optimization: a bilevel programming approach," *Structural and Multidisciplinary Optimization*, vol. 20, no. 3, pp. 214–221, 2000.

[8] A. Sinha, P. Malo, and K. Deb, "An improved bilevel evolutionary algorithm based on quadratic approximations," in *2014 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1870–1877.

[9] V. Oduguwa and R. Roy, "Bi-level optimisation using genetic algorithm," in *2002 IEEE International Conference on Artificial Intelligence Systems (ICAIS)*, 2002, pp. 322–327.

[10] S. D. Scott, A. Samal, and S. Seth, "HGA: A hardware-based genetic algorithm," in *Proceedings of the 1995 ACM third international symposium on Field-Programmable Gate Arrays*. ACM, pp. 53–59.

[11] L. Guo, D. B. Thomas, C. Guo, and W. Luk, "Automated framework for FPGA-based parallel genetic algorithms," in *the 24th International Conference on Field Programmable Logic and Applications (FPL)*, Sept 2014, pp. 1–7.

[12] L. Guo, C. Guo, D. B. Thomas, and W. Luk, "Pipelined genetic propagation," in *IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2015.

[13] D. B. Thomas and W. Luk, "High quality uniform random number generation using lut optimised state-transition matrices," *The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 47, no. 1, pp. 77–92, 2007.

[14] D. B. Thomas and W. Luk, "An FPGA-specific algorithm for direct generation of multi-variate gaussian random numbers," in *Application-specific Systems Architectures and Processors (ASAP)*, 2010, pp. 208–215.

[15] A. Sinha, P. Malo, and K. Deb, "Efficient evolutionary algorithm for single-objective bilevel optimization," *arXiv preprint arXiv:1303.3901*, 2013.