A Parallel Hardware Architecture For Quantum Annealing Algorithm Acceleration

A Parallel Hardware Architecture For Quantum Annealing Algorithm Acceleration

Evelina Forno*, Andrea Acquaviva*, Yuki Kobayashi†, Enrico Macii*, and Gianvito Urgese*
* Politecnico di Torino, Torino, Italy, 0039 011 090 7042. Email: gianvito.urgese@polito.it
† NEC Corporation, Kawasaki, Japan. Email: y-kobayashi@hq.jp.nec.com

***Abstract*—Quantum Annealing (QA) is an emerging technique, derived from Simulated Annealing, providing metaheuristics for multivariable optimisation problems. Studies have shown that it can be applied to solve NP-hard problems with faster convergence and better quality of result than other traditional heuristics, with potential applications in a variety of fields, from transport logistics to circuit synthesis and optimisation. In this paper, we present a hardware architecture implementing a QA-based solver for the Multidimensional Knapsack Problem, designed to improve the performance of the algorithm by exploiting parallelised computation. We synthesised the architecture using as a target an Altera FPGA board and simulated the execution for solving a set of benchmarks available in the literature. Simulation results show that the proposed implementation is about 100 times faster than a single-thread general-purpose CPU without impact on the accuracy of the solution.**

I. Introduction

Optimization problems can be encountered in many fields of science and technology, from the synthesis of electronic circuits (Boolean satisfiability problems) to transportation and location logistics (Vehicle routing problem). In most cases, to solve such problems means to find the global optimum of a multivariable function, while fulfilling a set of constraints. Global optimization problems are NP-hard in complexity; as such, finding the exact solution is often unfeasible in terms of computation time. However, many heuristic methods have been developed to provide good quality, approximate solutions in a short time.

The list of available heuristic algorithms is long, and new variants are being developed every year. Among the most well-established [1] we find *Simulated Annealing* (SA) [2]. Inspired by the physical process of thermal annealing in materials science, SA is controlled by tuning the value of a temperature parameter T over a given period of time. The algorithm randomly generates moves in the search space that are always accepted when the cost function H is improved; moves that would result in the cost increasing are accepted with a probability $e^{\frac{\Delta H}{T}}$. SA is well-tested, efficient and robust, but requires rather long computation times.

Simulated Annealing is widely used in the VLSI field for the generation of connection paths, placement and other optimization. For example, it has been successfully employed in solving floorplanning [3] and placement [4] problems, which are classified as NP, obtaining better results than other types of heuristics such as PSO and Ant Colony. An outstanding example is the use of SA in placement algorithms within commercial software such as Quartus II [5] for FPGA design optimization. Research within the field has led to several improvements to SA, especially in parallelization efforts, which yield better results and linear speedups with respect to the classic algorithm while trying to balance the added complexity in synchronizing several solver processes.

Quantum Annealing is an emerging technique derived from Simulated Annealing. Previous papers have demonstrated that for some problems (such as the Traveling Salesman Problem (TSP) [6], the Multidimensional Knapsack Problem (MKP) [7], and the Ising spin glass [8]), QA has a faster convergence in terms of simulation steps, providing results of improved quality.

However, Quantum Annealing is even more computationally expensive than SA when executed on standard PCs. As such, there has been interest from several research teams in developing parallel hardware architectures for accelerating SA on FPGAs and GPU. Since SA is an inherently sequential algorithm, workarounds are necessary to exploit concurrency; solutions have been proposed both on CPU [4] [5] and GPU [9] which run independent SA solvers in separate threads, then choose the best solution reached among all threads. A similar approach has been attempted on FPGA [10], reporting success for relatively small problems (1024-bit, corresponding to a 32-city TSP problem). FPGA acceleration of Monte Carlo solvers, which apply concepts compatible with SA, has found great success in physical simulations of the nearest-neighbor Ising spin glass such as the *Janus II* computer [11]; however, most NP problems require higher levels of connectivity.

Regarding the Multidimensional Knapsack Problem, implementations have been made on GPU for heuristic solvers [12] [13], reporting speedup over parallel CPU solutions. Exact solvers [14] were realized on GPU for the 0-1 Knapsack Problem, but literature suggests that exact MKP solvers may be too demanding for current GPU architectures.

In this paper, we propose the design of a hardware architecture implementing a solver for the Multidimensional Knapsack Problem using *Quantum Annealing* (QA). We analyzed the behavior of the Quantum Annealing algorithm and applied modifications to improve its performance. We also parallelised computation with a multi-core architecture and applied architectural and functional optimizations to reduce calculation times. We described the QA solver in a High Level Synthesis

language and synthesized it with an *Altera Stratix V FPGA* as target. The results in this paper are derived from RTL simulation.

The rest of the paper is organized as follows: in Section II we provide a mathematical description of the Quantum Annealing algorithm. In Section IV we discussed our proposed architecture, whereas we reported the performances achieved by our accelerator in Section IV. Finally we give some conclusions (Section V).

II. Background

Quantum Annealing is a metaheuristic algorithm inspired by classical Simulated Annealing. Instead of applying a thermal gradient to the system, it applies a slowly diminishing *transverse field*.

The quantum annealing process can be simulated in a traditional computer using stochastic techniques like the Monte Carlo method. This algorithm involves an adaptation of the classical Metropolis-Hastings algorithm [8] to step out of local optimum solutions.

What sets apart Quantum Annealing from Simulated Annealing is the emulation of the *quantum tunneling* effect for escaping local minima. The key parameter Γ, which indicates the strength of the transverse field, represents the quantum tunneling width and determines the radius of local search. At first, the neighborhood comprises the whole search space; during the annealing this radius is gradually reduced.

A benchmark for QA [8] is the *Ising spin glass*, a mathematical model of ferromagnetism used in statistical mechanics. This model consists of a system of up/down spins organized in a graph. Each spin has a given radius of neighbor spins that it is allowed to interact with. The model is described by the Hamiltonian reported in equation 1.

$$H_c = -\sum_{\langle i,j \rangle} J_{ij} s_i s_j \tag{1}$$

Where the N spins s_i can take the values ± 1. The interaction between spins s_i and s_j on lattice sites i and j is described by the exchange coupling J_{ij}. $\langle i,j \rangle$ means that i and j are neighbor spins; the radius of neighborhood depends on the chosen model. When spins are not neighbors their interaction is $J_{ij} = 0$, therefore such pairs do not contribute to the Hamiltonian.

The Monte Carlo simulation of the Ising spin glass consists of iterating over every spin and performing an update, i.e., deciding whether or not to flip each spin based on the status of its neighbors and the strength of its interaction with them. A Monte Carlo step is concluded when all the spins in the systems have been updated.

To perform QA of Ising spin glasses, an additional term is added to the Hamiltonian by applying a transverse magnetic field Γ, as shown in equation 2.

$$H_q = -\sum_{i<j}^{N} J_{ij}\sigma_i\sigma_j - \Gamma(t)\sum_{i}^{N}\sigma_i \tag{2}$$

Γ starts out at a high value and is gradually reduced to zero during the annealing.

In computer-simulated QA [8] [7] [6], the quantum effect is simulated by mapping the partition function of the quantum Ising model to that of a classical Ising model in one higher dimension, called *imaginary time* dimension or *Trotter* dimension (as the Suzuki-Trotter expansion is used to perform this mapping). This means that the system is simulated simultaneously in R different iterations or *replicas* (Fig. 1), which start out completely independent but have a correlation factor J_t that grows in time, forcing them to converge to a single solution at the end of annealing. J_t is calculated each MCS as a function of Γ, by the formula in equation 3.

$$J_t = -\frac{1}{2} log(tanh(\Gamma)) \tag{3}$$

QA is not just SA with R copies running in parallel. Normally, SA is only able to pass to a neighboring state on the energy landscape in one step, by thermal transitions. However, by adding the J_t coupling, QA is able to tunnel through energy barriers, avoiding local maxima, and exploring the state space more effectively. This can explain the faster convergence of Quantum Annealing.

In particular, thermal transition probability is proportional to $e^{\frac{-\Delta}{k_B T}}$ (where Δ is the height of the energy barrier, k_B is the Boltzmann constant, and T the annealing temperature). This probability is dominated by the height Δ of the barrier, which means it is difficult to get out of a very deep well of local minimum by means of thermal fluctuations. However, it has been demonstrated [15] that the quantum tunneling probability through the same barrier is proportional to $e^{\frac{-\sqrt{\Delta} w}{\Gamma}}$.

The tunneling probability depends not only on the height Δ, but also on the width w of the energy barrier. This means that QA shows significant advantage on problems where the energy landscape presents a high amount of perturbation with many high and thin barriers ($w \ll \Delta$). Indeed, the search of the ground state for an Ising spin glass model is one of these problems: since many NP problems can be formulated through

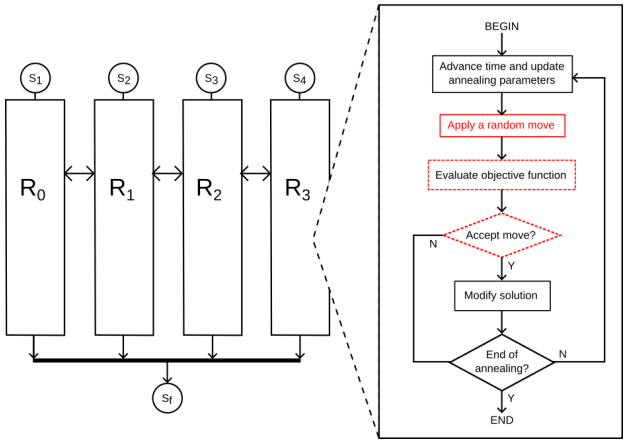

Fig. 1: The Quantum Annealing algorithm. On the left, a visual representation of replicas as a series of parallel threads exchanging information with neighbors about their local solution. On the right, the flowchart of the algorithm executed within each replica: highlighted in red are the portions of the program that can be rewritten to fit different problem types (e.g., to allow only legal moves within the constraint system). The red dashed line indicates parts that stay the same, save for the problem Hamiltonian.

the Ising model [16], we can apply QA to them and expect favorable results.

III. Implementation

The architecture (shown in Fig. 2) is composed of an array of R processors, each representing a Trotter replica for Quantum Annealing. Each replica shares its current $itemvector$ with its neighbors.

There is also a *controller* module that ensures synchronization of the replicas during one Monte Carlo step (MCS). It fetches the J_t value for the current MCS from memory, then enables the replicas to allow calculation of the next step. It also receives the current *total profit* from each replica and detects when a new optimum has been found.

Finally, we have a *Restricted Quantum Annealing (RQA) engine* that receives the item vectors from the processors and calculates the frequency of each item across all solutions. It outputs a binary vector describing the locked items that replicas are no longer allowed to change. In the following we will examine each module in detail.

We described this circuit behaviorally in SystemC and perform high-level synthesis with the NEC *CyberWorkBench HLS compiler* (CWB) [17], exporting the components to an RTL format with the same procedure described in [18].

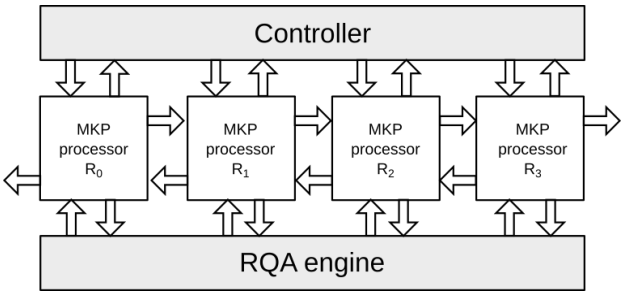

Fig. 2: Block diagram of the FPGA architecture

A. MKP Processor

The basic structure of the MKP processor is the same for all replicas. The core of this processor is a Finite State Machine that executes the operations for one Monte Carlo step of Quantum Annealing. Every MKP processor stores a copy of the problem data in local registers.

The processor is a Finite State Machine of 23 states for the largest problem (30x500); because of a few branches in the algorithm, the average latency due to pipelining is lower than 23 cycles. However, since each random number generation attempt using the LFSR introduces an extra cycle of latency, the overall latency of this module is *not deterministic*.

Indeed, a key problem of implementing a stochastic algorithm on FPGA is the quality of the Random Number Generator (RNG). We use a simple 32-bit LFSR which provides good pseudorandom performance. From this LFSR we select up to 9 bits as an item identifier (*itemRNG*) and 16 bits for the Metropolis random number (*metroRNG*). When the MKP solver needs a random number, it enables the LFSR and waits for the next clock cycle. Since the LFSR is a synchronous circuit, it necessarily introduces latency.

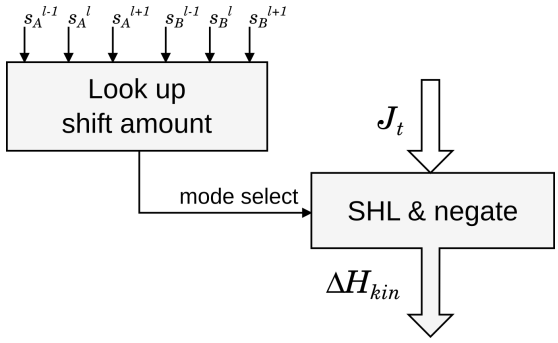

Fig. 3: Block diagram of the J_t multiplication stage

Most of the necessary instructions in the process datapath are adders, subtractors, and comparators. However, when we enter a Metropolis attempt to swap item A with item B, the calculation of the quantum portion of the Hamiltonian:

$$\Delta H_{kin} = J_t \cdot ((s_A^{l-1} + s_A^{l+1}) \cdot 2s_A^l + (s_B^{l-1} + s_B^{l+1}) \cdot 2s_B^l) \quad (4)$$

would introduce at least one 16-bit multiplier. We observe that the result of the right hand side parenthesis only has a few discrete possible values: $-8, -4, 0, +4, +8$. Then, the ΔH_{kin} calculation can be accomplished by using exclusively LUTs and shifters, as shown in Fig. 3. This saves a considerable amount of area and reduces the critical path.

The calculation of the exponential $e^{-\Delta H}$ for the Metropolis trial is also prohibitively expensive in FPGA, so we implement it with a LUT indexed by ΔH_{kin}. We experimentally determined that a high precision is not necessary for this operation and the exponential LUT only needs 24 entries of 16 bits.

When it is necessary to evaluate ΔH_{kin}, each replica needs to have a stable copy of the item vectors from its neighbors, and these vectors should be from the same annealing step that the replica is currently in. FPGA implementations of the Ising model [19] solve this problem by partitioning the spins into two groups and sharing each spin unit between two neighboring spins that are processed in separate clock cycles.

From simulation statistics we determine that calculation of ΔH_{kin} is not performed in 98-99% of annealing steps within a simulation. Then, the replicas can indeed work in parallel most of the time. In our final implementation we enable all the replicas at once; because of the low granularity of our moves, a replica can use a neighbor's result from the previous MCS, accepting a possible error of at most 2 bits. We confirm that the quality of result is equivalent to the one with the partitioned replicas. By allowing all replicas to process at the same time, we improve overall latency by about 50%.

The processor core implements the QA algorithm for MKP proposed by Bergé *et al.* [7]. We apply two modifications:

1) The mutation proposed in [7] is to try exchanging an item *a* outside the bag with an item *b* inside the bag. If the exchange is not possible, "step back" by removing item *b* from the bag. After making several trials, we observed that by avoiding to "step back", we are able to obtain better results.
2) The value of J_t plays a paramount role in the annealing. However, no prescription for it is given in [7]. The available benchmark problems present a very broad range

$(10 - 1000)$ of possible values for the item prices, and therefore a broad range of possible values for the potential energy ΔH_{pot}. Since stepping out of the local optimum depends on the value of $\Delta H = \Delta H_{pot} + \Delta H_{kin}$, it is evident that ΔH_{kin} must be roughly of the same order of magnitude as ΔH_{pot} for the Metropolis dynamics to work. ΔH_{kin} is directly proportional to J_t, therefore we want J_t to be of the same order of magnitude as the range of item prices. One possible way to change the value of J_t is to change the value of Γ_0: however, modifying Γ_0 also influences the rate of growth of J_t. By trial and error we identified the ideal rate of growth as that corresponding to $\Gamma_0 = 3.0$, as lower values cause the J_t interaction to spike to high values too early in the annealing (preventing the system to explore the state space for most of the annealing), while high values make the growth of J_t too slow (greatly reducing the quantum tunneling effect). The method we settle on is to "amplify" J_t to fit the given problem set: before the annealing, we scan the matrix of item prices and calculate the average value. Then we use that to multiply J_t throughout the annealing. This allows us to improve results significantly, especially in problems that have high values for prices, like the ones in the Chu-Beasley benchmark.

B. Controller

The role of the controller is to keep replicas synchronized over the same Monte Carlo step. It stores the 16-bit values for J_t in a memory block of τ entries. When all the replicas are done updating, the controller disables them and fetches the next J_t value. Then the replicas are enabled and a new MCS begins. Once all J_t values have been read it raises the *QA_done* output signal and stops the annealing.

C. RQA Engine

The Restricted Quantum Annealing (RQA) engine's role is to keep track of the frequency of appearance of each item across solutions. We accomplish this by means of a SWAR algorithm for popcount (or Hamming weight counter), which is essentially a series of $\log(R) + 1$ sum, right shift and bit masking steps.

For each item i, the input to the popcount is built out of a vertical slice of all the replicas' item vectors, taking from each only the ith bit (as highlighted in red in Fig. 4). The result of the popcount is stored in a *frequency vector* at position i. If the frequency is greater than the RQA *blocking frequency*, the ith bit of the output signal *lockedItems* is set.

Using popcount lets us avoid computing long sums with the 500-position item vectors of the hardest benchmark. Every vertical slice we build is R bits long, which is generally much shorter than the item vectors.

Additionally, we can explore the design space for this component by performing loop unrolling to control the quantity of items processed at the same time and the pipeline depth of the frequency counter. We found that the smallest-area, smallest-delay, longest-latency implementation is the most efficient: we process only one item at a time, and we allow CWB's automatic scheduling to reduce the delay as much as possible, resulting in a $(\log(R) + 1)$-cycle latency.

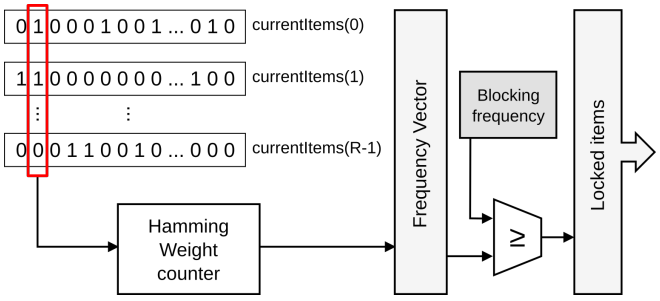

Fig. 4: Block diagram of the RQA engine

Although the replicas don't have access to the most updated version of the *lockedItems* vector at all times, we verified in simulation that this minimized implementation has no adverse effects on the speed of convergence of RQA or the quality of result. Then, we can add RQA to the system with negligible impact on area and maximum frequency.

IV. Results and Discussions

We performed RTL synthesis in CWB with the Altera Stratix V (5SGXEA7N2F45C2) FPGA as target. We then performed RTL simulation of the resulting Verilog files to extract the timing performance results reported in this section. The Verilog files were then input to Altera Quartus Prime for logic synthesis.

A. Area and Critical Path

In Tab. I are reported the logic synthesis results for a system with $R = 16$, $\tau = 1000000$, 500 items and 30 constraints.

The OR30x500 family of benchmarks is the largest one available to us and the one we considered for final implementation. Meanwhile, the block memory occupation also depends on τ. We synthesized up to 1 million steps, which is an appropriate simulation time for the 30x500 problem instances.

The total area of logic utilization grows linearly with the number of replicas. This is to be expected since every replica added corresponds to a new processor core in the architecture.

Examining the growth of area when varying the number of constraints and items shows that area also grows linearly with the problem size.

We compared the maximum frequency estimated by Quartus Prime over a wide range of synthesis results corresponding to various combinations of number of replicas, number of items, and number of constraints.

From Fig. 5, it is evident that the worst-case maximum frequency is decreasing as the area grows. We can identify at least two main reasons for this. i) combinational paths become

Logic utilization	147,236 / 234,720 (63 %)
Total registers	157782
Total block memory bits	18,457,600 / 52,428,800 (35 %)
Maximum frequency	164 MHz
Total Thermal Power Dissipation	8.559 mW

TABLE I: Results of QA hardware architecture logic synthesis

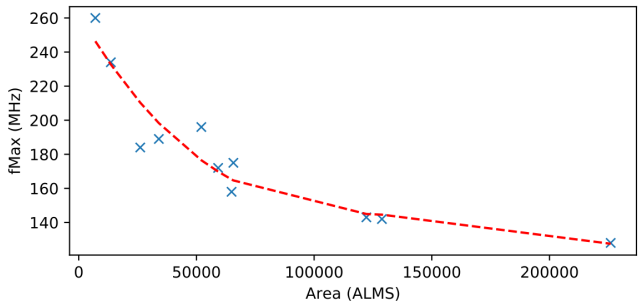

Fig. 5: Maximum frequency obtained by Quartus synthesis for varying hardware size.

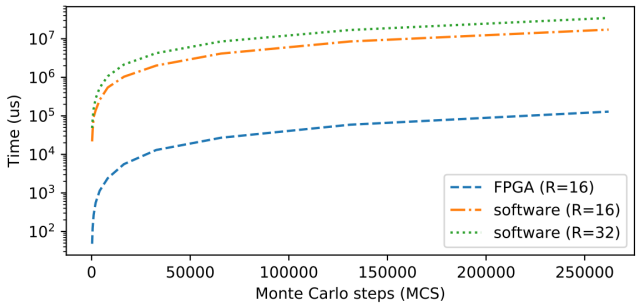

Fig. 7: Execution times for the FPGA and software versions for the benchmark OR30x500-0.25.

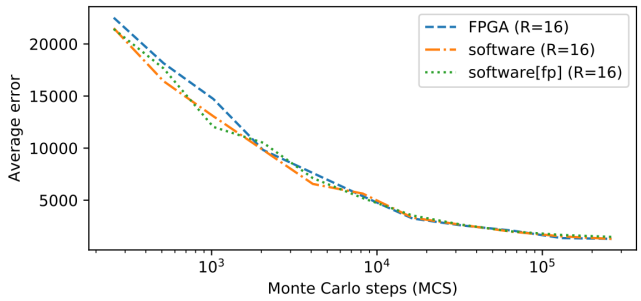

Fig. 6: Comparison of results from the FPGA and software versions of the algorithm for the benchmark OR30x500-0.25, R = 16. Quality of result is not lost in the transition from floating point (fp) to fixed point data representation.

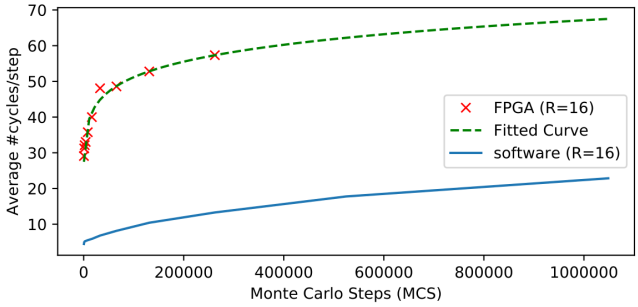

Fig. 8: Comparison of the growth in number of random trials per step as the annealing time increases, benchmark problem OR30x500-0.25. The regression function used is $f(\lambda) \approx 13.17\lambda^{0.12}$.

longer and slower as the width of parameters grows, especially of the item vector. ii) interconnections also become longer and more complex, causing increased delays. In practice, increasing the number of replicas or the size of the problem makes the system slower.

B. Quality of results compared to CPU version

We created a bit-compatible integer model in software to estimate performance of an FPGA version ahead of implementation. As shown in Fig. 6, the behavior is completely coherent with the floating point version, displaying similar convergence across a wide range of τ.

Then, it is fair to compare the behavior of the FPGA implementation (in RTL simulation) with the floating-point software version of the algorithm.

C. QA-HW vs QA-CPU

The execution time for different values of τ is charted in Fig. 7. At the maximum frequency of 164 MHz, the FPGA solution is much faster than the software at executing the same number of Monte Carlo steps. However, in the FPGA version the number of cycles per step appears to increase quite strongly as τ increases; while the speedup over software is about $350\times$ at τ = 2048, at τ = 32768 it is only $210\times$, and this advantage will probably continue to decrease for greater values of τ. This problem may be attributed to the LFSR's limited ability to generate an adequate number.

How much does the RNG problem affect the FPGA latency? We compared the average latency per step of the FPGA implementation with the average number of RNG trials in the software version. We suppose that the two parameters are directly proportional as the FPGA latency per step is predominantly determined by the number of LFSR trials, with little (and generally constant) overhead from the rest of the algorithm. Fig. 8 shows how the latency due to the RNG trials, after a sharp initial growth, settles into a slowly increasing curve, compatible with that encountered in the software simulation.

Then, we expect that the FPGA implementation's speedup will maintain a reasonable advantage ($150-180\times$ faster) over the software version even as we increase the annealing time.

Finally, in Fig. 9 we show the speed of convergence to result for the two versions of the algorithm. It is clear that the hardware version can produce a similar quality of result as the software version in less computation time.

Our FPGA implementation appears to compare favorably with GPU results reported in literature; the parallelised QA solver reports lower computation time than the Ant Colony Optimization on GPU from [12] (408 ms for the largest instance, rather than $\sim$ 10000 ms). We reserve comparison

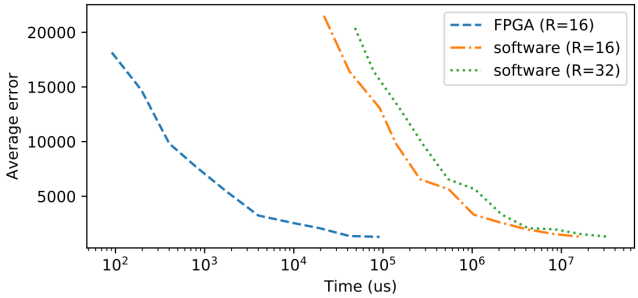

Fig. 9: Speed of convergence to result for the software and FPGA versions, benchmark problem OR30x500-0.25.

with a GPU implementation of QA for future work.

Though synthesis provides limited information on the power consumption, it is still possible to make an optimistic estimation of the energy savings. The software version ran on a computer with an Intel i7 920 CPU; it has a maximum I_{cc} per core equal to 145 A and a typical associated V_{cc} per core of 0.131 V. Assuming our single-thread software ran on a single core at full load, that would mean an instantaneous power of 18.995 W; since the program ran for 17.319 s, the estimated energy consumption is about 329 J. Meanwhile, the FPGA energy consumption is around 8.559 W $\times$ 0.108 s = 0.924 J, leading to estimated energy savings of $\sim 350\times$.

V. Conclusion

In this paper we propose a parallel architecture for the solution of the Multidimensional Knapsack Problem using a Quantum Annealing solver. We implemented our hardware architecture in a High Level Synthesis language and synthesized on an FPGA target. The parametric design instantiates multiple computation cores, synchronized by a Controller module that regulates the annealing process. We also implemented an optional module for computing a Restricted Quantum Annealing solution, allowing to improve the quality of result with a negligible cost in terms of area and latency.

Simulation shows that our QA solver provides the same quality of result as a floating point software version, while outperforming a single-thread CPU. We can demonstrate analytically that the QA solver is at least $150\times$ faster than software. In addition, synthesis reports show that the implementation has a reasonable logic utilization even for the largest problems.

Future works will focus in further improving and testing the FPGA architecture. As it is, the memory containing the problem parameters is implemented in the FPGA logic elements; the block memory utilization can be improved by offloading the data to external memory. We plan to further improve the architecture and continue with the placement on an FPGA board to evaluate the real performance of the proposed accelerator.

We will also develop a CPU-GPU multithread implementation of the QA solver in order to compare the effective speedup and power savings for the FPGA version.

Acknowledgments

The HLS compiler and the technical support were provided by NEC Corporation, Japan.

References

[1] Beheshti and Shamsuddin. "A Review of Population-based Meta-Heuristic Algorithm". In: *Int. J. Advance. Soft Comput. Appl., Vol. 5, No. 1, March 2013* (2013).

[2] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by Simulated Annealing". In: *Science* 220.4598 (1983), pp. 671–680. ISSN: 00368075, 10959203. URL: http://www.jstor.org/stable/1690046.

[3] Jenifer J, Anand S, and Levingstan Y. "SIMULATED ANNEALING ALGORITHM FOR MODERN VLSI FLOORPLANNING PROBLEM". In: 2 (Apr. 2016), pp. 175–181.

[4] Atanu Roy Karthik Ganesan Pillai. "Parallel Simulated Annealing for VLSI Cell Placement Problem". In: 2009.

[5] Adrian Ludwin and Vaughn Betz. "Efficient and Deterministic Parallel Placement for FPGAs". In: *ACM Trans. Design Autom. Electr. Syst.* 16 (2011), 22:1–22:23.

[6] Roman Marto ňák, Giuseppe E. Santoro, and Erio Tosatti. "Quantum annealing of the traveling-salesman problem". In: *Phys. Rev. E* 70 (5 Nov. 2004), p. 057701. DOI: 10.1103/PhysRevE.70.057701. URL: https://link.aps.org/doi/10.1103/PhysRevE.70.057701.

[7] P. Bergé et al. "Restricting the search space to boost Quantum Annealing performance". In: *2016 IEEE Congress on Evolutionary Computation (CEC)*. July 2016, pp. 3238–3245. DOI: 10.1109/CEC.2016.7744199.

[8] Roman Marto ňák, Giuseppe E. Santoro, and Erio Tosatti. "Quantum annealing by the path-integral Monte Carlo method: The two-dimensional random Ising model". In: *Phys. Rev. B* 66 (9 Sept. 2002), p. 094203. DOI: 10.1103/PhysRevB.66.094203. URL: https://link.aps.org/doi/10.1103/PhysRevB.66.094203.

[9] Y. Han, S. Roy, and K. Chakraborty. "Optimizing simulated annealing on GPU: A case study with IC floorplanning". In: *2011 12th International Symposium on Quality Electronic Design*. Mar. 2011, pp. 1–7. DOI: 10.1109/ISQED.2011.5770735.

[10] Sanroku Tsukamoto et al. "An Accelerator Architecture for Combinatorial Optimization Problems". In: 2017.

[11] M. Baity-Jesi et al. "Janus II: A new generation application-driven computer for spin-system simulations". In: *Computer Physics Communications* 185.2 (2014), pp. 550–559. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2013.10.019. URL: http://www.sciencedirect.com/science/article/pii/S0010465513003470.

[12] Henrique Fingler et al. "A CUDA based Solution to the Multidimensional Knapsack Problem Using the Ant Colony Optimization". In: *Procedia Computer Science* 29 (2014). 2014 International Conference on Computational Science, pp. 84–94. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2014.05.008. URL: http://www.sciencedirect.com/science/article/pii/S1877050914001859.

[13] Bianca de Almeida Dantas and Edson Norberto Cáceres. "Sequential and Parallel Implementation of GRASP for the 0-1 Multidimensional Knapsack Problem". In: *Procedia Computer Science* 51 (2015). International Conference On Computational Science, ICCS 2015, pp. 2739–2743. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.05.411. URL: http://www.sciencedirect.com/science/article/pii/S1877050915012193.

[14] M. E. Lalami and D. El-Baz. "GPU Implementation of the Branch and Bound Method for Knapsack Problems". In: *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*. May 2012, pp. 1769–1777. DOI: 10.1109/IPDPSW.2012.219.

[15] Arnab Das, Bikas K. Chakrabarti, and Robin B. Stinchcombe. "Quantum annealing in a kinetically constrained system". In: *Phys. Rev. E* 72 (2 Aug. 2005), p. 026701. DOI: 10.1103/PhysRevE.72.026701. URL: https://link.aps.org/doi/10.1103/PhysRevE.72.026701.

[16] Andrew Lucas. "Ising formulations of many NP problems". In: *Frontiers in Physics* 2 (2014), p. 5. ISSN: 2296-424X. DOI: 10.3389/fphy.2014.00005. URL: https://www.frontiersin.org/article/10.3389/fphy.2014.00005.

[17] Kazutoshi Wakabayashi. "CyberWorkBench: Integrated design environment based on C-based behavior synthesis and verification". In: *VLSI Design, Automation and Test, 2005.(VLSI-TSA-DAT). 2005 IEEE VLSI-TSA International Symposium on*. IEEE. 2005, pp. 173–176.

[18] Marco Bettoni et al. "A convolutional neural network fully implemented on fpga for embedded platforms". In: *CAS (NGCAS), 2017 New Generation of*. IEEE. 2017, pp. 49–52.

[19] Francisco Ortega-Zamorano et al. "FPGA Hardware Acceleration of Monte Carlo Simulations for the Ising Model". In: *CoRR* abs/1602.03016 (2016). arXiv: 1602.03016. URL: http://arxiv.org/abs/1602.03016.