A Mixed Precision Methodology for Mathematical Optimisation

Gary C.T. Chow and Wayne Luk
Department of Computing
Imperial College London
London SW7 2AZ, United Kingdom
{cchow, wl}@doc.ic.ac.uk

Philip H.W. Leong
School of Electrical and Information Engineering
University of Sydney
Sydney, Australia
philip.leong@sydney.edu.au

***Abstract*—This paper introduces a novel mixed precision methodology for mathematical optimisation. It involves the use of reduced precision FPGA optimisers for searching potential regions containing the global optimum, and double precision optimisers on a general purpose processor (GPP) for verifying the results. An empirical method is proposed to determine parameters of the mixed precision methodology running on a reconfigurable accelerator consisting of FPGA and GPP. The effectiveness of our approach is evaluated using a set of optimisation benchmarks. Using our mixed precision methodology and a modern reconfigurable accelerator, we can locate the global optima 1.7 to 6 times faster compared with quad-core optimiser. The mixed precision optimisations search up to 40.3 times more starting vector per unit time compared with quad-core optimisers and only 0.7 to 2.7 % of these searches are refined using GPP double precision optimisers. The proposed methodology also allows us to accelerate problems with more complicated functions or solve problems involving higher dimensions.**

***Keywords*-mathematical optimisation, FPGA, reconfigurable, mixed-precision, Nelder-Mead downhill simplex**

I. Introduction

Unconstrained mathematical optimisation involves minimising an objective function $f(\vec{x}) : \mathbb{R}^k \longrightarrow \mathbb{R}$ by selecting an input vector $\vec{x}_0$ such that $\forall \vec{x} \in \mathbb{R}^k, f(\vec{x}_0) \leq f(\vec{x})$. Mathematical optimisation is extensively used in many areas such as engineering [8, 9], economics [7], finance [6], circuit design [1, 5], real time control [11], statistics and machine learning [4, 10], to name just a few. Solving a mathematical optimisation can be time consuming, especially for high-dimensional non-convex problems with multiple local minima. The performance issue becomes a major obstacle for time-critical problems such as finance and real-time control applications.

The implementation of an optimiser varies with the objective function, constraints, dimension, parameters and the choice of the optimiser algorithm. Hence, it is impractical to accelerate mathematical optimisation with application specific integrated circuits (ASICs). Reconfigurable hardware such as Field Programmable Gate Arrays (FPGAs) becomes a viable hardware option because they can be reconfigured to adapt to different parameters.

The ability to support customizable data-paths of different precisions is an important advantage of FPGAs. Reduced-precision data-paths usually have higher clock frequencies and consume fewer resources. Given the same amount of FPGA resources and execution time, we can handle problems with more complicated cost function or higher dimension using reduced-precision data-paths. However, reduced-precision data-paths may introduce computation errors which direct the search to produce poor results. Thus there are trade-offs between quality of the optimisation result, precision of data-paths, and parallelism of the optimisation process. When high quality optimisation results are required, high-precision data-paths are unavoidable.

This paper investigates the trade-offs associated with reduced precision arithmetic in FPGA based mathematical optimisers. We aggressively reduce the precision to realise FPGA based simplex optimisers with high performance and parallelism. The reduced precision optimisers filter and select potential regions that might contain optima. We then verify these regions using high precision optimisers running on conventional General Purpose Processors (GPPs). The mixed precision methodology also exploits the synergy between FPGA and GPP, by allowing them to work in precisions that they are specialised at. Mixed precision methodologies for other application domains are proposed in [2, 3].

II. A parameterisable simplex optimiser algorithm

Algorithm 1 shows the Nelder-Mead downhill simplex used in our FPGA optimisers. The algorithm attempts to locate the nearest local optimum by maintaining a set of test vectors (i.e. $\vec{x}_i$) which are referred to as the simplex. For an objective function with k input variables, the simplex is a set of $(k + 1)$ vectors in $\mathbb{R}^k$. In each iteration, the cost of each vector in the simplex is evaluated and the best, the second best and the worst vectors are marked. Using the reflection, expansion and the contraction processes (steps 5-18), up to three new test vectors are generated. If any of them are better than the worst vector, it is accepted to replace the worst vector. If none of them have a lower cost, it is assumed that a better point might lie within the simplex and hence all vectors except the best are shrunk towards the best vector (step 19). The simplex algorithm continues iteratively until the entire simplex is smaller than a certain tolerance ϵ (steps 21 to 23), or the maximum number of iterations has been reached.

Algorithm 1 The Nelder-Mead downhill simplex method.

Input: k = the dimension of the problem. $\{\vec{x}_i\}$ = a set of (k+1) initial vectors in $\mathbb{R}^k$. ϵ = the x tolerance. N_{max} = maximum iteration steps. $\vec{lb}, \vec{ub} \in \mathbb{R}^k$, lower and upper boundaries of the inputs. f = the objective function.

Scaling coefficients:

$\alpha = 1$, $\gamma = 2$, $\beta = 0.5$ and $\delta = 0.5$

```
1:  loop
2:    $count \leftarrow 0$
3:    evaluate $f(\vec{x}_1), f(\vec{x}_2) \cdots f(\vec{x}_{k+1})$
4:    set $f_h = \max(f_i)$ , $f_s = \max_{i \neq h}(f_i)$, $f_l = \min(f_i)$
5:    $\vec{x}_0 = \frac{1}{k}\sum_{i \neq h} \vec{x}_i$, /* the centroid of all but the worst
      points in the simplex */
6:    $\vec{x}_r \leftarrow \vec{x}_0 + \alpha(\vec{x}_0 - \vec{x}_h)$ /* reflection */
7:    if $f_l \leq f_{x_r} \leq f_s$ then
8:      $\vec{x}_h \leftarrow \vec{x}_r$
9:    else if $f_{x_r} < f_l$ then
10:     $\vec{x}_e \leftarrow \vec{x}_0 + \gamma(\vec{x}_0 - \vec{x}_h)$ /* expansion */
11:     $f_{x_e} < f_{x_r}$ ? $\vec{x}_h \leftarrow \vec{x}_e$ : $\vec{x}_h \leftarrow \vec{x}_r$
12:   else
13:     if $f_s \leq f_{x_r} \leq f_h$ then
14:       $\vec{x}_c \leftarrow \vec{x}_0 + \beta(\vec{x}_0 - \vec{x}_h)$ /* outside contraction */
15:       $f_{x_c} < f_{x_r}$ ? $\vec{x}_h \leftarrow \vec{x}_c$ : goto shrink_phase
16:     else if $f_{x_r} > f_h$ then
17:       $\vec{x}_c \leftarrow \vec{x}_0 - \beta(\vec{x}_0 - \vec{x}_h)$ /* inside contraction */
18:       $f_{x_c} < f_{x_h}$ ? $\vec{x}_h \leftarrow \vec{x}_c$ : goto shrink_phase
19:     end if
20:     shrink_phase: $\vec{x}_{i\_new} = \vec{x}_l - \delta(\vec{x}_l - \vec{x}_i)$ for $i =
        1 \cdots (k+1)$ and $i \neq l$
21:   end if
22:   $count++$
23:   if $\vec{x}_i - \vec{x}_l < \epsilon$ for all $i \neq l$ or $count > N_{max}$ then
24:     break and return $\vec{x}_l$ and $f_{x_l}$
25:   end if
26: end loop
```

Figure 1 shows the system architecture of the reconfigurable simplex optimiser. It consists of a new vector generation unit, which can generate a reflection, an expansion or a contraction vector every clock cycle. A fully parallelised implementation is used for the $f(x)$ evaluation data-path. To solve the data dependency problem, we execute P simplex optimisation instances with the same cost function and different starting points in parallel. Instead of using conventional registers, we use circular registers with loop lengths equal to P in our $\vec{x}_i$ and $f(x_i)$ tracking units. Thus, every simplex optimisation instance is served by the hardware every P clock cycles. If P is larger than the latency of the entire data-path, the pipeline is always full

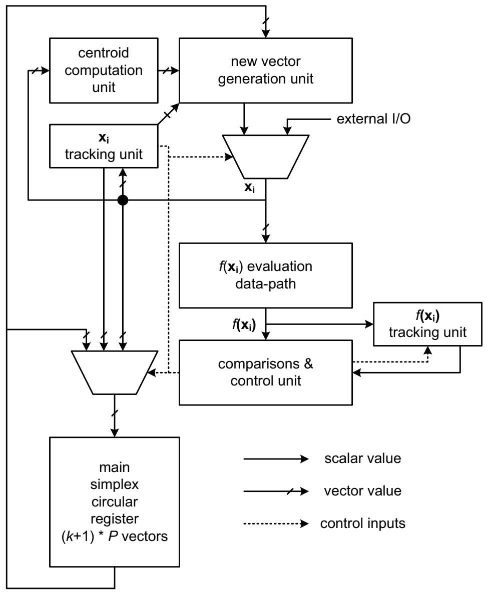
centroid computation unit
new vector generation unit
x_i tracking unit
external I/O
x_i
f(x_i) evaluation data-path
f(x_i)
f(x_i) tracking unit
comparisons & control unit
main simplex circular register (k+1) * P vectors
scalar value
vector value
control inputs

Figure 1. System architecture of the reconfigurable simplex optimiser.

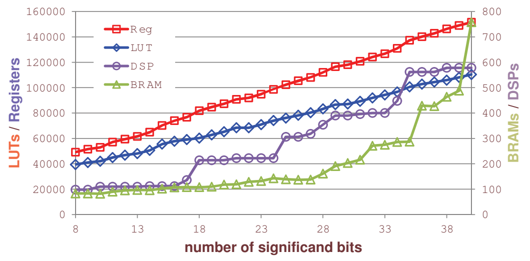

Figure 2. Cost of the simplex optimiser with different number of significand bit for solving the Ackley function with dimension = 12.

and the optimiser does not need to be stalled. The required lengths of the circular registers are usually in the order of tens to hundreds and they can be realised efficiently using FPGA block memories. Figure 2 shows the cost of the FPGA simplex optimisers with different number of significand bits for solving the Ackley function with dimension $k = 12$. The cost advantage of using reduced precision datapaths are clearly shown in the figure.

III. Mixed Precision Basin Hopping Optimisation

Algorithm 2 shows the mixed precision basin hopping methodology in our reconfigurable accelerator system. In each iteration, P_{search} new starting vectors are generated using mutation from the current global vector. Coarse-grained optimisation results are generated by the FPGA reduced precision simplex optimisers. The best few starting vectors with cost difference smaller than a certain refinement margin ϵ_{margin} are re-optimised using double precision optimisers running on GPP and the results are compared with the current best solution. Hence, the FPGA optimisers are only used to select and filter potential starting vectors

Algorithm 2 Mixed precision parallel basin hopping optimisation.

Input: f = the objective function.
P_{search} = number of parallel reduced precision simplex search in each iteration.
ϵ_{margin} = refinement margin.
N = number of iteration.
$\vec{x}_0$ = initial vector.
ϵ = tolerance of GPP optimiser.
Processes:
$\vec{x}_{new} \leftarrow \mathcal{M}(\vec{x})$, vector mutation process.
$f_i \leftarrow \mathcal{S}_{fpga}(\vec{x}_i, N_{max}, L)$, FPGA simplex run with initial vector $\vec{x}_i$ and N_{max} iteration in precision L.
$(f_{best}, \vec{x}_{best}) \leftarrow \mathcal{S}_{gpp}(\vec{x}_i, \epsilon)$, GPP simplex run with initial vector $\vec{x}_i$ and tolerance ϵ in reference precision.

```
1:  $f_{global} \leftarrow f(\vec{x}_0)$; $\vec{x}_{global} \leftarrow \vec{x}_0$
2:  for $iter = 1 \rightarrow N$ do
3:      /* Parallel FPGA searching */
4:      for $i = 1 \rightarrow P_{search}$ do
5:          $\vec{x}_i \leftarrow \mathcal{M}(\vec{x}_{global})$
6:          $f_i \leftarrow \mathcal{S}_{fpga}(\vec{x}_i, N_{max}, L)$
7:      end for
8:      sort $f_i$ in ascending order
9:      /* GPP refinement */
10:     for $i = 1 \rightarrow P_{search}$ do
11:         if $f_i - f_i < \epsilon_{margin}$ then
12:             $(f_{best}, \vec{x}_{best}) \leftarrow \mathcal{S}_{gpp}(\vec{x}_i, \epsilon)$
13:             if $f_{best} < f_{global}$ then
14:                 $f_{global} \leftarrow f_{best}$; $\vec{x}_{global} \leftarrow \vec{x}_{best}$
15:             end if
16:         end if
17:     end for
18: end for
```

for GPP optimisers.

Parameter selection One assumption of our empirical technique is that the optimisation parameters can be selected based on knowledge of a similar problem. For example, a family of the Schwefel function with different constant offsets should share similar characteristics. Although such an assumption seems restrictive, it is realistic because similar optimisation problems can be solved by FPGA without regenerating a new bitstream. We suggest the following steps to select the parameters:

1) N_{max} - Although setting N_{max} to a higher value decreases the average optimisation error μ_{opt_error}, the FPGA optimisers are merely for selecting starting vectors which potentially lie in the basin containing the global optimum. Hence, we can set N_{max} to a relatively small value. Experimental results show that setting $N_{max} = 0.25 \times N_{avg}$ works well for all the benchmarks we tested. ϵ using random starting vectors and use N_{avg} as N_{max}.
2) L - precision of the FPGA optimiser. If L is high, the cost of each FPGA optimiser is increased. In extreme cases, we might not even be able to place a single optimiser into the FPGA. On the other hand, if L is low, optimisation errors will increase and the refinement margin ϵ_{margin} needs to be increased, leading to more workload for the GPPs. We suggest to start the search for the optimal precision from mid-range (such as 25 significand bits) and try both sides until the best is found.
3) P_{search} - once the precision L is fixed, we can estimate the maximum number of simplex optimisers fitted into the FPGA using the resources matrix. The parallelism of the basin hopping algorithm P_{search} should be fixed to $P \times P_c$, where P is the minimum pipelined stage of each FPGA optimiser and P_c is the number of optimisers fitted into the FPGA.
4) ϵ_{margin} - the refinement margin should be large enough such that the starting vector with the global optimum is not missed by the GPP re-optimisation due to optimisation error and yet it is not too large to stress the GPP. Experimental results show that the optimal value for ϵ_{margin} usually varies between 1-2 times of μ_{opt_error}, where μ_{opt_error} is the average optimisation error profiled using random starting vectors and $N_{max} = 0.25 \times N_{avg}$.
5) evaluation - a user should evaluate the mixed precision basin hopping at this point, and go back to step 2 to choose another reduced precision if required.

IV. Results

We use the MaxWorkstation reconfigurable accelerator system from Maxeler Technologies for our experiments. It has a MAX3424A card with a Xilinx Virtex-6 SX475T FPGA. The card is connected to an Intel i7-870 GPP through a PCI express link and the GPP has 4 physical cores. The MaxWorkstation allows us to design a parameterisable simplex optimiser such that the dimension, cost function, constraints and precision of the floating point data-paths can be changed at compile time. For all our implementations, we use a floating point representation with 8-bit exponents and different number of significand bits. To evaluate our mixed precision methodology, we select three common optimisation benchmarks as shown in Table I. These are usually considered difficult cases since they have multiple local optima but only one global optimum.

Figure 3 shows the average execution time and the required number of GPP double precision simplex refinements for the Ackley benchmark function with parameters selected using the method proposed in Section III. As we can see from the figure, the optimal FPGA reduced precision for the problem is $s17e8$. To the left of the optimal precision,

benchmark	lb_i	ub_i	cost function
Ackley	-32.768	32.768	$f(\vec{x}) = -20exp\left(-0.2\sqrt{\frac{1}{k}\sum_{i=1}^{k} y_i^2}\right) - exp\left(\frac{1}{k}\sum_{i=1}^{k}(cos2\pi y_i)\right) + 20 + e$
Rastrigin	-5.12	5.12	$f(\vec{x}) = 10k + \sum_{i=1}^{k}(y_i^2 - 10cos(2\pi y_i))$
Schwefel	-500	500	$f(\vec{x}) = 418.9829k - \sum_{i=1}^{k}(y_i sin\sqrt{\lvert y_i\rvert})$
$\vec{y} = \vec{x} - \vec{c}$, where $\vec{c}$ is a constant offset vector between 0 to 1			

Table I
BENCHMARKS PROBLEMS USED FOR EVALUATION, DIMENSION k IS SET TO 12.

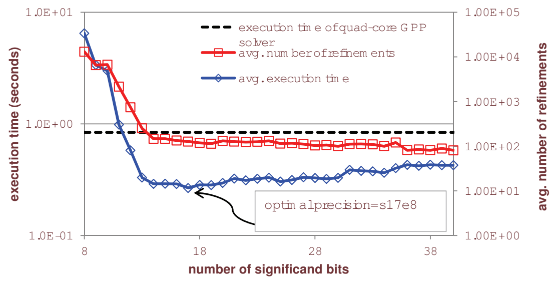

Figure 3. Average required execution time and the required number of GPP double precision simplex refinement to locate the global optimum for the Ackley benchmark function. 50 trial runs are used.

the refinement margin ϵ_{margin} is large so many GPP double precision refinements are required which offsets the execution time reduction due to higher FPGA clock frequency and parallelism. To the right of the optimal precision, the number of refinement is small but the cost of FPGA optimisers is high. A similar pattern is also observed in other benchmark problems. It is interesting to note that although the optimal FPGA reduced precision is $s17e8$, the proposed mixed precision methodology provides performance gains over a wide range of precisions ($s12e8$ to $s40e8$). Hence it is not necessary for us to pick the optimal reduced precision as long as it is not too far away from the optimum.

Numerical experiments are run on the three benchmarks and we observed 1.7 to 3.6 times performance gains on the reconfigurable accelerator using our mixd precision methodology against quad-core GPP implementations. The effectiveness of our mixed precision methodology is attributed to the abilities of the FPGA optimisers to search huge amount of starting vectors in parallel. The mixed precision methodology search 40.3 times more starting vector per unit time compared with quad-core GPP optimisers. Only 0.7 to 2.43 % of these searches need to be refined by GPP double precision optimisers.

V. CONCLUSION

This paper proposes a novel mixed precision methodology for mathematical optimisation problems in reconfigurable accelerator systems. It exploits the synergy between the FPGA and GPP and produces results identical to double precision solvers running on GPP. Experimental results on three common optimisation benchmark problems show that our methodology allows up to 40.3 times more searches per unit time compare with optimisers for the quad-core GPPs. Using the proposed methodology and a modern reconfigurable accelerator system, we can locate the same set of global optima 1.7 to 3.6 times faster on average and search 40.3 times more starting vectors compared with quad-core GPP implementations.

Acknowledgments

The research leading to these results has received funding from the Croucher Foundation, Maxeler Technologies, Xilinx, the UK EPSRC, and the European Union Seventh Framework Programme under grant agreements number 257906 and 248976.

REFERENCES

[1] S. P. Boyd, S.-J. Kim, D. D. Patil, and M. A. Horowitz. Digital circuit optimization via geometric programming. *Operations Research*, 53:899–932, 2005.
[2] G. Chow, K. Kwok, W. Luk, and P. Leong. Mixed precision processing in reconfigurable systems. In *Proc. FCCM*, pages 17 –24, may 2011.
[3] G. C. T. Chow, A. H. T. Tse, Q. Jin, W. Luk, P. H. Leong, and D. B. Thomas. A mixed precision monte carlo methodology for reconfigurable accelerator systems. In *Proc. FPGA*, pages 57–66, 2012.
[4] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.
[5] M. del Mar Hershenson, S. P. Boyd, and T. H. Lee. Optimal design of a CMOS Op-Amp via geometric programming. *IEEE Trans. CAD*, 20:1–21, 2001.
[6] H. Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):pp. 77–91, 1952.
[7] A. Mas-Colell, M. D. Whinston, and J. R. Green. *Microeconomic Theory*. Oxford University Press, USA, 1995.
[8] S. S. Rao. *Engineering Optimization: Theory and Practice*. Wiley, 2009.
[9] J. N. Siddall. *Optimal Engineering Design (Dekker Mechanical Engineering)*. CRC Press, 1982.
[10] V. Vapnik. *The Nature of Statistical Learning Theory (Information Science and Statistics)*. Springer, 1999.
[11] Y. Wang and S. Boyd. Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2):267–278, 2010.