Hardware implementation of the DFP algorithm using inexact line search

Kaikai Su¹, Anhua Guo²
¹² College of Physics & Information Engineering of FuZhou University, FuZhou, FuJian P.R. China
*Corresponding author’s e-mail: n191127112@fzu.edu.cn

Abstract. The Quasi-Newton method (QN) is widely used in solving large-scale optimization problems due to its high efficiency. However, this algorithm requires a large number of iterative calculations, which is more time-consuming to compute in software. The Davidon-Fletcher-Powell (DFP) algorithm has been implemented on field-programmable gate array (FPGA) in order to speed up the computation. The number of iterations in the online search part of the algorithm is excessive, which affects the speed of the calculation. In order to solve this problem, this paper proposes a hardware implementation of DFP algorithm using inexact linear search. The results show that the calculation speed of the DFP algorithm implemented in this paper on the software side and FPGA platform is increased by 7.59 times and 3.08 times, respectively. And resource consumption and accuracy have little effect on the results.

1. Introduction
Optimisation methods are a general term for the study of disciplines that seek to find certain factors in order to optimise an indicator[1]. The most common optimization methods are gradient descent, Newton, Quasi-Newton and conjugate gradient, etc. Among them, the Quasi-Newton method is one of the most effective methods for solving nonlinear optimization problems. The essential idea of the Quasi-Newton method is to improve the shortcoming of the Newton method that it needs to calculate the inverse of the Hessian matrix every time. The Quasi-Newton method uses a positive definite matrix to replace the inverse of the Hessian matrix, thus simplifying the operation[2]. However, optimization methods require constant iterative calculations, which may be time-consuming if run on a computer. FPGA has always been regarded as an effective tool for accelerating calculations. In recent years, more and more algorithms have been implemented on FPGA.

At present, many optimization methods are implemented through FPGA, and the calculation speed is greatly improved. Sun has implemented genetic algorithms in different ways on FPGA, which are widely used in applications with high applicability[3]. Liu implemented the BFGS algorithm on FPGA and improved the matrix update module. The results show that the resource consumption has been drastically reduced [4]. James proposed to use the Quasi-Newton method for image restoration. Comparing multiple algorithms, the results show that the Quasi-Newton method has the best effect[5]. Liu has proposed a method for implementing DFP on FPGA. The results show that the proposed hardware implementation of the design can achieve up to 17 times acceleration compared to the software implementation [6]. However, the line search in the DFP algorithm takes up a lot of time. To solve this problem, this paper proposes an improvement on this basis.

Therefore, the DFP algorithm is implemented on FPGA in this paper. The design is divided into multiple calculation modules, making full use of the advantages of FPGA that can be calculated in
parallel to accelerate the calculation of the DFP algorithm. And on this basis, in order to speed up the computation, this design uses inexact line search algorithm instead of exact line search algorithm. The algorithm is then implemented on an FPGA to further improve the speed of the DFP algorithm’s computation.

This paper is organized as follows. In the second section, the DFP algorithm and the inexact line search algorithm are briefly introduced. In the third part, the implementation process of DFP algorithm and inexact line search algorithm on FPGA is described in detail. In the fourth section, the implementation results of the algorithm on hardware and software are analyzed. The fifth section concludes this design.

2. Materials and Methods

Newton method is an unconstrained objective optimization method using the first derivative and the second derivative. Its basic idea is to update the search direction in the Newtonian direction at each iteration. However, the Newton method requires a more stringent requirement for the derivability of the objective. And the algorithm needs to calculate the inverse matrix of the Hesse matrix, which is more complicated to calculate. In order to solve this problem, the DFP algorithm was proposed by Davidon in 1959. Through this method, the calculation amount of Newton method is greatly reduced.

The specific steps of the DFP algorithm are as follows:

DFP algorithm

Step 0: Initialization. Pick the initial point \( w_0 \in R^n \), if \( ||g_0|| = 0 \), the algorithm terminates. Otherwise, set \( H_0 = I \) and \( k = 0 \).

Step 1: Calculate search direction \( d_k = -H_k g_k \).

Step 2: Calculate step size \( \lambda_k \) by the formula: \( f(w_k + \lambda_k d_k) = \min_{\lambda \geq 0} f(w_k + \lambda_k d_k) \).

Step 3: \( w_{k+1} = w_k + \lambda_k d_k \), if \( ||g_{k+1}|| = 0 \), the algorithm terminates. otherwise, go to step 4.

Step 4: \( s_k = w_{k+1} - w_k \), \( y_k = g_{k+1} - g_k \). Calculate \( H_{k+1} \) by the formula: \( H_{k+1} = H_k - \frac{y_k s_k^T}{y_k^T H_k y_k} + \frac{s_k s_k^T}{y_k^T s_k} \).

Step 5: \( k = k + 1 \), go to step 1.

In step 2, the method used for the search step is the one-dimensional exact search method - the Golden Section method. In the calculation process of the DFP algorithm, the golden section method calculates the objective function many times. The online search occupies more than half of the calculation time in the entire calculation process of DFP.

For the Quasi-Newton Method, the speed of convergence does not depend on the exact one-dimensional search process. The disadvantage of exact line search is that it is inefficient when the initial point of the iteration is very far from the solution of the problem. However, this problem can be well solved by inexact line search, the main idea of which is to compute a step size in the obtained search direction that gives a suitable amount of descent to the objective function. In this way the solution can be approached more quickly. This kind of algorithm does not require that the step length of each iteration of the objective function be minimized, but it makes the algorithm drop rapidly in a macroscopic view. Compared to exact line search, inexact line search requires less computation. A classical method of inexact line search, the Armijo-Goldstein, is described below.

Armijo-Goldstein

Step 0: Initialization. Enter the values of \( W_k, d_k, g_k \). Pick the initial point \( \lambda_0 \in (0,0.5) \), \( t > 1 \). Set \( a_0 = 0 \), \( b_0 = 2 \), \( k = 0 \).

Step 1: If \( E_T(W_k + \lambda_k d_k) \leq E_T(W_k) + \rho \lambda_k g_k^T d_k \), go to step 2; Otherwise, set \( a_{k+1} = a_k, b_{k+1} = \lambda_k, \) go to step 3.

Step 2: If \( E_T(W_k + \lambda_k d_k) \geq E_T(W_k) + (1 - \rho) \lambda_k g_k^T d_k \), the algorithm terminates and takes \( \lambda_k \) as the result; Otherwise, set \( a_{k+1} = a_k, b_{k+1} = b_k \). If \( b_k < 2 \), go to step 3; Otherwise, set \( \lambda_{k+1} = t \lambda_k, k = k + 1 \), go to step 1.

Step 3: \( \lambda_{k+1} = \frac{a_k + b_k}{2}, k = k + 1 \), go to step 1.
3. Hardware implementation

The overall framework diagram of DFP based on FPGA platform is shown in Figure 1. The whole framework diagram includes gradient update module, matrix update module, line search module and objective function module.

![Figure 1: Framework diagram of the DFP algorithm using inexact line search](image)

As shown in Figure 1, the gradient module is a gradient update module, which is used to calculate the gradient calculation required for each iteration. Matrix update is a matrix update module that calculates the new matrix needed for each iteration by correcting the formula for the matrix. The Function module is the objective function module, which is used to obtain the function values needed for the calculation process. The objective function chosen in the design of this paper is:

\[ f(\omega) = (\omega_1 - \omega_2)^2 + (\omega_3 - \omega_4)^2 + (\omega_5 - 1)^2 + (\omega_6 - 2)^2. \]

The AG-Line module is an inexact line search module, using the Armijo-Goldstein algorithm, which is used to calculate the update step size for each iteration. Both the line search module and the gradient calculation module need to calculate the objective function. In order to save resources, the design idea of module reuse is adopted. The linear search and the gradient calculation share a function calculation module. The DP in the diagram is a vector multiplication module, implemented by combining a multiplier and an accumulator. The whole design makes full use of FPGA’s advantages of pipeline and parallelism to speed up the calculation of DFP algorithm.

This design replaces the exact line search algorithm with inexact line search algorithm - Armijo-Goldstein. The diagram of the Armijo-Goldstein architecture implemented on FPGA is shown in Figure 2. When the module starts to run, the module will first calculate \( E_T(W_k) \) for each iteration calculation. In the step of calculating \( E_T(W_k + \lambda_k d_k) \leq E_T(W_k) + \rho \lambda_k g_k^T d_k \), both sides of the inequality are calculated simultaneously. If the output sel1 of the comparator is 1, \( E_T(W_k + \lambda_k d_k) \geq E_T(W_k) + (1 - \rho)\lambda_k g_k^T d_k \) will be calculated next. Otherwise, a, b and \( \lambda \) start to update. The update formula for \( \lambda \) is \( \lambda_{k+1} = (a_{k+1} + b_{k+1})/2 \). If the result of the comparator sel2 is 1, the module will end and output \( \lambda \) as the result. If the result of the comparator sel2 is 0, a and b and \( \lambda \) start to be updated. At this time, if b=2, set \( \lambda_{k+1} = 2\lambda_k \), otherwise set \( \lambda_{k+1} = (a_{k+1} + b_{k+1})/2 \). In this design, the initial values of a and b are 0 and 2 respectively, and the initial values of \( t \) and \( \lambda_0 \) are 2 and 1 respectively. Throughout the calculation, the adder and multiplier used to update \( \lambda_{k+1} \) are shared with the adder and multiplier used to calculate \( W_k + \lambda_k d_k \) in order to save resources.
4. Experimental results

The proposed design is synthesized and implemented on the FPGA (Xilinx Artix-7 XC7A200T) board, using Xilinx VIVADO 2018.03. The clock frequency of this design is 250MHz. This article mainly compares and evaluates from two aspects of resource consumption and running time.

In terms of resource consumption, the overall resource consumption of the DFP algorithm using exact line search and the DFP algorithm using inexact line search are given in Table 1. Table 2 shows the resource consumption of each module of the DFP algorithm using exact line search. The resource consumption of each module of the DFP algorithm using inexact line search is given in Table 3. The results show that the difference in resource consumption between the DFP algorithm using exact line search and the DFP algorithm using inexact line search is not significant. Due to the different calculation methods of the line search module, the DFP algorithm that uses inexact line search will take up more resources. A comparison of Tables 2 and 3 shows that there is a difference in resource consumption between the line search modules of the two algorithms. The reason for this result is that the inexact line search module uses an extra DP module, which causes a slight increase in resource consumption. However, this has no impact on the overall design.

| Table 1 | Comparison of the resource consumption of the two algorithms |
|---------|-------------------------------------------------------------|
|          | FF   | LUT  | LUTRAM | DSP |
| exact line search | 24601 | 14726 | 872   | 114 |
|             | (9.19%) | (11%) | (1.89%) | (15%) |
| inexact line search | 25335 | 15074 | 918   | 121 |
|             | (9.47%) | (11.27%) | (1.99%) | (16.35%) |

| Table 2 | Resource consumption of DFP using exact linear search |
|---------|----------------------------------------------------------|
|          | FF  | LUT  | LUTRAM | DSP |
| gradient | 1638 | 912  | 19     | 7   |
| golden section | 2815 | 1757 | 51     | 14  |
| function  | 3772 | 2254 | 104    | 33  |
| matrix update | 10373 | 5704 | 475    | 34  |
Table 3 Resource consumption of DFP using inexact linear search

|                  | FF   | LUT  | LUTRAM | DSP |
|------------------|------|------|--------|-----|
| gradient         | 1638 | 921  | 19     | 7   |
| AG-line          | 3430 | 2056 | 101    | 21  |
| function         | 3772 | 2073 | 104    | 33  |
| matrix update    | 10373| 5854 | 475    | 34  |

This paper gives the comparison of the running times of the DFP algorithm using exact line search and the DFP algorithm using inexact line search on software and on FPGA respectively. Both algorithms were calculated using the same data. The software version of the DFP method is implemented in MATLAB. The software program was compiled with MATLAB 2018 and executed on r7-4800u CPU at 1.8 GHz with 16GB of RAM. Table 4 shows the total calculation time and calculation error (average error) of the two algorithms implemented on FPGA and software respectively. The results show that the DFP algorithm implemented in this paper is 3.08 times faster than the DFP algorithm implemented on the FPGA platform using exact line search, and 7.59 times faster than the DFP algorithm implemented in software using exact line search. In terms of error, the higher accuracy of the calculation on the software is due to the different representation of the two platforms, but it does not affect the results of the calculation.

Table 4 total computation time and error of the DFP algorithm implemented on different platforms.

|                          | time      | error    |
|--------------------------|-----------|----------|
| exact line search- software | 74325ns   | 0.00012  |
| exact line search-FPGA    | 29448ns   | 0.00063  |
| inexact line search- software | 61083ns    | 0.00022  |
| inexact line search-FPGA  | 97888ns   | 0.00078  |

5. Conclusion:
This paper improves the line search part of the DFP algorithm, and proposes to replace the exact line search algorithm with inexact line search algorithm, and finally implements the DFP algorithm on FPGA. This design makes full use of the FPGA’s advantages of pipeline and parallel computing to improve the calculation speed of the DFP algorithm. The results show that the calculation speed of the DFP algorithm implemented in this paper is increased by 7.59 times and 3.08 times on the software side and on the FPGA platform, respectively. And the resource consumption and accuracy rate do not affect the results.

Reference:
[1] Mykel J. (2019) Algorithms for Optimization. MIT Press, Massachusetts.
[2] Wei, Z, et al. (2005) New quasi-Newton methods for unconstrained optimization problems. Applied Mathematics and Computation, 175:1156-1188.
[3] Sun, X, et al. (2019) Design of FPGA Hardware based on Genetic Algorithm. In: 3rd International Conference on Computer Engineering, Information Science & Application Technology (ICCIA 2019). Chongqing,113-119.
[4] Liu, J and Liu, Q. (2018) Resource Reduction of BFGS Quasi-Newton Implementation on FPGA Using Fixed-Point Matrix Updating. In: 2018 28th International Conference on Field Programmable Logic and Applications (FPL) 2018, pp. Dublin, Ireland. pp. 301-3015.
[5] James G. Nagy and Katrina Palmer. (2004) Quasi-Newton methods for image restoration. In: SPIE Optics + Photonics. Denver, Colorado, United States. 412-422.
[6] Liu, Q, et al. (2016) FPGA-based acceleration of Davidon-Fletcher-Powell Quasi-Newton optimization method. Transactions of Tianjin University, 22:381-387.