Portfolio Optimization

Niels Stchedroff
Riskcare, London, e-mail: niels.stchedroff@riskcare.com

Abstract
Portfolio optimization is computationally intensive and has potential for performance improvement. This paper examines the effects of evaluating large numbers of proposed solutions in parallel for use with direct search optimization. This leads to a method that has considerable performance increase. A GPU implementation demonstrates an order of magnitude performance improvement over the best available multi-threaded CPU. The new algorithm and GPU combined offer a performance increase of more than 60× compared with CPU.

Keywords
GPU, CUDA, portfolio, optimization, multi-directional, covariance

Introduction
Assuming we have a portfolio consisting of n risky stocks, no risk-free asset, and (positive definite) VCV matrix σ, our objective is to find the vector of portfolio weights w that minimizes the total portfolio variance. We therefore need to find the global minimum of the objective function:

$$f(w) = V(w, \sigma) = w' \sigma w \quad (1)$$

where the prime denotes a transpose. The constraint we impose is that the portfolio weights must sum to one, i.e., $\mathbf{1}'w = 1$, where $\mathbf{1}$ denotes a vector whose elements are all equal to 1.

This paper examines the possibilities for increasing computational efficiency by evaluating multiple values for w in parallel. This is achieved by applying direct search techniques to the optimization.

Direct search methods
Direct search (the general class of optimization methods of which multi-directional [2] optimization is an example) has undergone a renaissance in the last two decades [1]. This is down to a better understanding of the mathematics underpinning the Nelder–Mead [3] and the multi-directional methods. They become effective when the problem is sufficiently complex or computationally difficult such that a perfect answer is unachievable [1].

Direct search methods do not rely on the objective function's derivatives or numerical approximation. The function to be optimized may be a "black box," with only the input parameters and the resultant value exposed to the optimization method [4].

The effectiveness of quasi-Newton methods and the availability of software tools that ease their use has caused direct search techniques to be overshadowed [1,5]. However, direct search techniques are invaluable [1] for problems that meet the following conditions:

- Calculation of $f(x)$ is very expensive or time-consuming.
- Exact first partial derivatives of $f(x)$ cannot be determined.
- Numerical approximation of the gradient of $f(x)$ is expensive in terms of time/resources.
- $f(x)$ is noisy, due to source data or the nature of the algorithm.

Two direct search methods of interest are the multi-directional search (MDS) method of Torczon [4,6,7] and the Nelder–Mead method [4,5]. Both MDS and Nelder–Mead are simple to code and analyze. It has been found that the Nelder–Mead algorithm (unlike MDS) can converge to non-minimizers when the dimensions of the problem are large enough [5,7].

For this reason, MDS was selected as the algorithm of choice for this work.

Both methods use an $n \times n$ matrix (simplex). This represents a starting point. Each row in the simplex represents the n input values for the function in question. The start point is randomly generated, within the required constraints for the input variables.

The best set of input values (i.e., the ones giving the smallest result) in the simplex is located and used to generate possible new values for each variable. Several different simplices are generated and evaluated against each other, as a simple step in the algorithm. The best is taken as the input for the next iteration.

This process of finding the best and generating a new simplex from it is continued until it is judged that an optimum value has been found. This means that for each iteration, n sets of test values must be evaluated a number of times.

MDS in detail

$$S = (v_1 .. v_n) \quad (2)$$

where S is the simplex. $v_0, \ldots, v_n$ are the vertices, where a vertex is defined as a vector of length n representing one possible set of portfolio weights, where v_0 is the best vertex found so far.

Three trials are made – reflection (R), expansion (E), and contraction (C).

$$R = (r_1 .. r_n) \quad (3)$$

where

$$r_i = v_0 - (v_i - v_0) \tag{4}$$

$$E = (e_1 .. e_n) \tag{5}$$

where

$$e_i = (1 - \mu) * v_0 + r_i \tag{6}$$

and μ is the *expansion* coefficient. A common default value is 2.0 [2].

$$C = (c_1 .. c_n) \tag{7}$$

where

$$c_i = (1 + \theta) * v_0 - \theta * r_i \tag{8}$$

and θ is the *contraction* coefficient. A common default value is 0.5 [2]. Then the updated simplex is computed as S_{next}.

If $R_{best} < f(v_0)$ and $E_{best} >= f(v_0)$
then $S_{next} = R$.
If $R_{best} < f(v_0)$ and $E_{best} < f(v_0)$
then $S_{next} = E$.
Otherwise, $S_{next} = C$.

Evaluation

We have seen in the previous section that we need to evaluate the objective function at n vectices for each evaluation of the simplex. This naturally leads to the suggestion that we consider a modified version where w is an $n \times n$ matrix itself. This can be represented in matrix format as below:

$$\begin{pmatrix} f_1 & .. & .. \\ .. & f_{..} & .. \\ .. & .. & f_n \end{pmatrix} = S' \sigma S \tag{9}$$

where

$$f_1 \cdots f_n$$

is the set of objective function values for the trial weights (vertices):

$$v_1 \cdots v_n$$

At first sight this might seem an inefficient method of calculation – we are throwing away:

$$(n \cdot n) - n$$

values in the final result. However, as we shall see, the performance issues are not obvious.

Weights

In the simplest formulation, the only constraint is that the weights sum to 1:

$$\sum_{k=1}^{n} w_k = 1.0 \tag{10}$$

The approach taken is to sum the actual weights, and use that value to scale the weights to 1.0. We can calculate the scale on the matrix of weights by multiplying by a vector of 1s:

$$\begin{pmatrix} v_1^T \\ v_{..}^T \\ v_n^T \end{pmatrix} * \begin{pmatrix} 1 \\ .. \\ 1 \end{pmatrix} \tag{11}$$

We can carry out the scaling by calculating the scaling factors with eq. (11), after each stage in the multi-directional algorithm.

Implementation

The problem has been reduced to two matrix multiplications, a matrix transposition, vector/matrix multiplication, and a vector/vector dot product multiplication. In the case of coding for a BLAS library, the transposition is handled by the settings for the DGEMM (or SGEMM) function. This means that for a BLAS, the core of the problem can be coded with two calls. The scaling operations are trivial in performance terms.

Various trials were run in developing this method. The discovery that the second matrix multiplication was more efficient in terms of time taken was a surprise–the original intention was to simplify implementation. The calculation times were tested using the Intel MKL BLAS library running on an i7 920 and on an NVIDIA C1060 GPU running the CUBLAS BLAS.

Of interest is the fact that little or no improvement is seen when running this calculation in Octave – which uses an unoptimized single-threaded version of the ATLAS BLAS for matrix operations. This strongly suggests that the performance gains are from the levels of optimization possible between matrix*matrix and matrix*vector calculations in the high-performance libraries (CUBLAS and Intel MKL) as shown in Figure 1. The initial very high

Figure 1: Performance improvement of the suggested method compared to using vector multiplications.

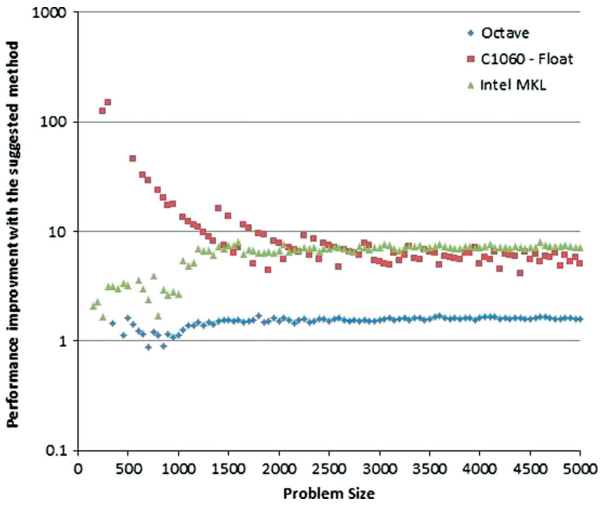
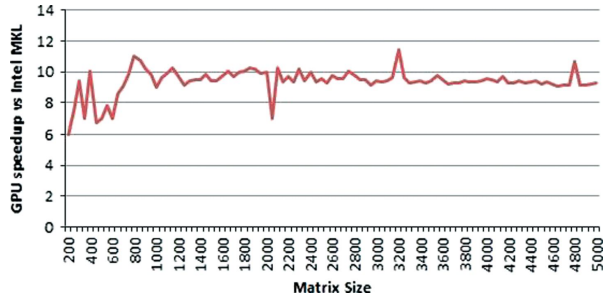


Figure 2: Performance of GPU vs. CPU for matrix*matrix BLAS operations.



gains for GPU are caused by the massive reduction in the number of kernels to be launched to complete the calculation.

The reason for the gain is that

$$T_s = 2 \cdot T_{mm} \tag{12}$$

$$T_o = n * (T_{vv} + T_{mv}) \tag{13}$$

$$T_s < T_o \tag{14}$$

for the high-performance BLAS options, where T_s is the time taken for the suggested method, T_o is the time taken for the original method, T_{mm} is the time taken for a matrix*matrix operation, T_{vv} is the time taken for a vector*vector operation, and T_{mv} is the time taken for a matrix*vector operation.

In terms of raw speed of GPU vs. CPU, the GPU is around nine times faster for computing matrix multiplications (see Figure 2). This is a severe test; the CPU is a high-end device, and the GPU is not the latest hardware. Many comparisons of GPU vs. CPU hardware use single-thread CPU code, whereas this case is a truly competitive comparison.

Applying multi-directional search on the GPU

With the potential performance outlined above, it seemed likely that the results of applying this evaluation method would be interesting. The MDS transformations were implemented on GPU, as well as the evaluation method outlined above. The selection of thenew best simplex was done on the CPU – a naturally single-thread function. All that the CPU is required to do is decide which path to follow – a comparison of two numbers.

GPU performance analysis

The scaling and MDS transformations were extremely efficient – they take on average less than 10% of the execution time for each iteration of the MDS algorithm. The other 90% is consumed by the evaluation. For example, for a 583 × 583 covariance matrix, the MD transformation took 0.135697 ms, the scaling 0.196647 ms, while the evaluation took 3.718785 ms.

The efficiency of the MD transformations on GPU boosts the overall performance vs. CPU to 10× (see Figure 3). Again, it should be noted that this is using a high-performance CPU with a top-notch BLAS library.

Figure 3: Performance of GPU vs. CPU for MDS.

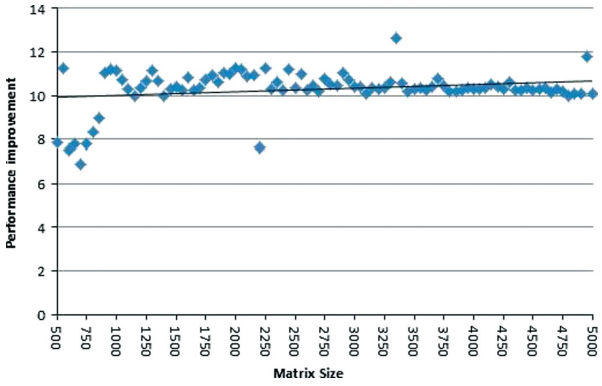
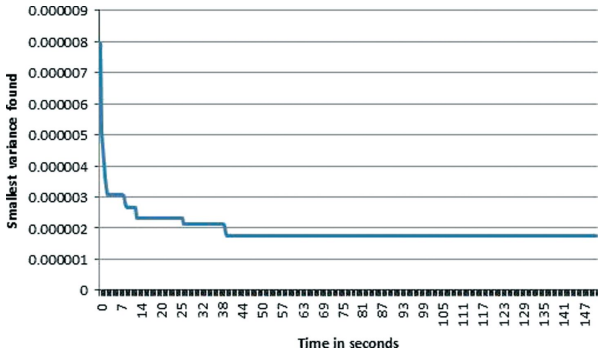


Figure 4: MDS performance on GPU for a 583-item portfolio.



The lower relative performance of GPU for portfolios smaller than 500 is due to the GPU not being fully loaded. If portfolios of this size are of particular interest, then an enhancement would be to compute the reflection, expansion, and contraction steps in parallel on the GPU. This would increase utilization on the GPU, bringing performance back to the 10× level.

Application in action

A variety of sizes of portfolio were run. For a 583-item portfolio (created from UK FTSE data), the minimum was reached in under 60 seconds, and the minimum variance for the weights selected was an order of magnitude smaller than that produced by a genetic algorithm optimizer. The MDS optimizer produced consistent results when started from different initial points (see Figure 4), showing a strong and rapid optimization. It was noticeable that the first iteration returned a result only 4 times greater than the final result in less than 1 second.

Returns vs. variance

Following the success of the methods outlined, when used for the simple portfolio optimization problem (1), the work was extended to include returns.

We define the vector of returns such that:

$$\mu = (\mu_1, \mu_2 \mu_n)^T \tag{15}$$

The mean return μ_ρ is given by

$$\mu_\rho = \mu^T w \tag{16}$$

where w is the vector of weights for the assets that we are evaluating. If the risk-free rate of return is r, then the Sharpe ratio is defined as

$$S = (\mu_\rho - r)/\sigma_\rho \tag{17}$$

where σ_ρ is the variance of return of the portfolio as calculated in eq. (1). Then the portfolio w^* with the optimal risk/return trade-off is the one with the highest Sharpe ratio, and is given by the solution to the following optimization problem:

$$\max S \tag{18}$$

$$w^T 1 = 1 \tag{19}$$

$$w_i >= lb_i \tag{20}$$

$$w_i <= ub_i \tag{21}$$

where 1 is a column vector of 1s, lb_i and ub_i denote the individual lower bound and upper bound, respectively. The bounds were implemented by giving a large penalty to sets of weights that exceeded them. The penalty is proportional to the amount that the given bounds are exceeded by. This establishes a trust region [8], which is highly effective in ensuring that the optimization algorithm tends toward solutions within the specified region.

Validation

When run for a 49-asset portfolio (picked from FTSE stocks randomly), the Sharpe ratio optimization reached a stable value in less than 3 seconds. The risk-free interest rate was taken as 0.5%.

The results were compared with a reference MATLAB implementation, using the *fmincon* optimization function (see Table 1).

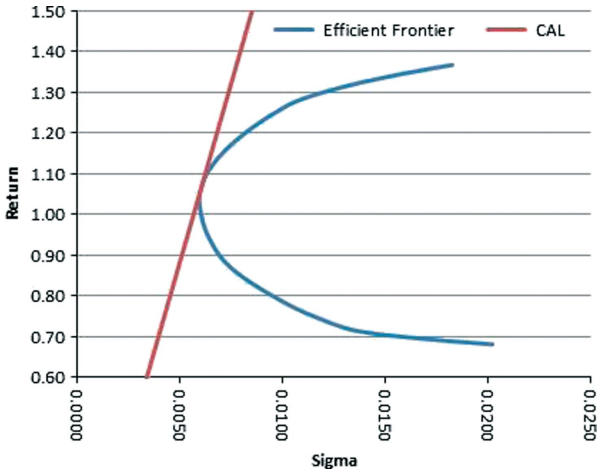
The efficient frontier was computed, and the CAL (capital allocation line) was plotted (see Figure 5). As expected, the results of the optimization matched the tangency portfolio values.

To evaluate performance for a large portfolio, an optimization was run on a 610-asset portfolio, selected from FTSE stocks. This achieved a stable Sharpe value in approximately 120 seconds. Each iteration took approximately 15 milliseconds.

Table 1: Sharpe ratio optimization results.

	GPU	MATLAB
Mean return (ρ)	1.068057	1.065769
σ	0.006045	0.006038
Sharpe	175.844	175.670008

Figure 5: Efficient frontier for the 49-asset portfolio.



Conclusions

A very noticeable speed-up has been obtained when evaluating trial values for the weights in blocks, for high-quality BLAS solutions. In addition, this method drastically simplifies the coding problem to a few standard calls to a BLAS library. This performance gain can be exploited by direct search optimization techniques.

Multi-directional search (MDS) was implemented and demonstrated as suitable for the task, with stable performance and high speed. When implemented on GPU, a 10× speed-up is seen, compared to a multi-threaded implementation of the same algorithm on a high-end CPU.

Combining GPU and the new algorithm gives a speed-up of more than 60× over a CPU running the simple line-by-line evaluation of the simplex – even with optimal multi-threaded code on the CPU.

Acknowledgments

Special thanks are due to my colleague Jimmy Law at Riskcare, who wrote the MATLAB reference implementation and provided help with the theoretical side of the work.

Niels Stchedroff is a Senior Consultant for Riskcare. He has a BSc in Computer Science from University College London and an MPhil in Operations Research from Southampton University.

REFERENCES

[1] M.H. Wright. 1996. Direct search methods: Once scorned, now respectable. In *Numerical Analysis 1995 (Proceedings of the 1995 Dundee Biennial Conference on Numerical Analysis)*, pp. 191–208. Addison Wesley Longman, Harlow, UK.

[2] V. Torczon. 1991. On the convergence of the multi-directional search algorithm. *SIAM Journal of Optimization*, 1:123–145.
[3] J.A. Nelder and R. Mead. 1965. A simplex method for function minimization. *Computer Journal*, 7:308–313.
[4] C.T. Kelley. 1999. *Iterative Methods for Optimization*. Society for Industrial and Applied Mathematics.
[5] R.M. Lewis, V. Torczon, and M.W. Trosset. 2000. Direct search methods: Then and now. *Journal of Computational and Applied Mathematics*, 124:191–207.
[6] E. Boyd, K.W. Kennedy, R.A. Tapia, and V.J. Torczon. 1989. Multi-directional search: A direct search algorithm for parallel machines. Technical report, Rice University.
[7] V. Torczon. 1989. *A Direct Search Algorithm for Parallel Machines*. PhD thesis, Rice University.
[8] Y. Yuan. 1999. A review of trust region algorithms for optimization. In *ICIAM 99: Proceedings of the Fourth International Congress on Industrial and Applied Mathematics*. Oxford University Press.

Book Club

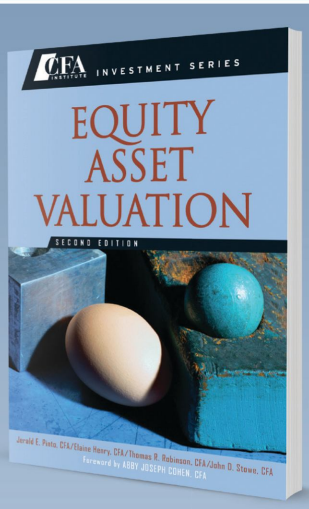
Share our passion for great writing – with Wiley's list of titles for independent thinkers ...

CFA INSTITUTE INVESTMENT SERIES

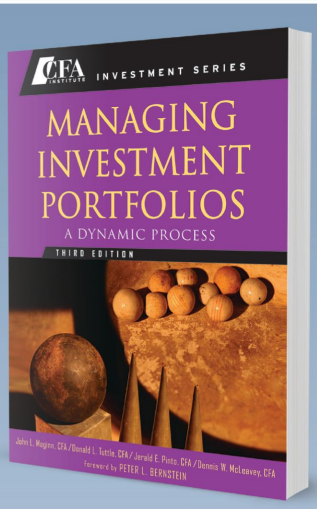
Turning students into professionals

Wiley Global Finance and the **Chartered Financial Analyst Institute®** are proud to present the **CFA Institute Investment Series**, bringing together highly regarded academics and financial professionals to create essential volumes on critical topics in finance.

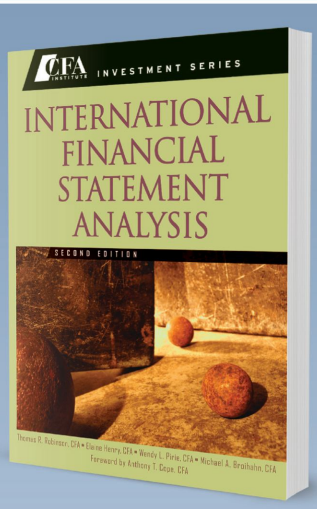
In each CFA Institute Investment Series book, thought leaders provide insight into theory and practice of important issues in finance. The books are ideal for the graduate-level finance student as well as the industry practitioner.



978-0-470-57143-9 • Hardback • 464 pages
March 2010 • ~~£65.00/€76.00~~ £39.00/€45.60



978-0-470-08014-6 • Hardback • 960 pages
April 2007 • ~~£70.00/€80.00~~ £42.00/€48.00



978-0-470-91662-9 • Hardback • 1040 pages
May 2012 • ~~£65.00/€76.00~~ £39.00/€45.60

To view the full series and access free content visit us at **www.wiley.com/go/cfa**

WILEY Global Finance
WHERE DATA FINDS DIRECTION

SAVE 40%

When you subscribe to *Wilmott* magazine you will automatically become a member of the **Wilmott Book Club** and you'll be eligible for 40 per cent discount on specially selected books in each issue when you order direct from **www.wiley.com** - just quote promotion code **WBC40** when you order.
The titles will range from finance to narrative non-fiction. For further information, call our **Customer Services Department** on **44 (0)1243 843294**, or visit wileyeurope.com/go/wilmott or wiley.com (for North America)