Genetic algorithms for discovery of matrix multiplication methods

A. Joó[1,2], A. Ekárt[1] and J. P. Neirotti[1]

[1]Aston University, Aston Triangle, Birmingham B4 7ET, UK

Email: {jooam, a.ekart, j.p.neirotti}@aston.ac.uk

[2]Sapientia University, 540485 Corunca, 1C Soseaua Sighisoarei, Romania

Abstract

We present a parallel genetic algorithm for finding matrix multiplication algorithms. For 3×3 matrices our genetic algorithm successfully discovered algorithms requiring 23 multiplications, which are equivalent to the currently best known human-developed algorithms. We also studied cases with fewer multiplications and found an approximate solution for 22 multiplications.

Matrix multiplication (MM) has numerous applications, where computational efficiency is crucial. The standard algorithm for multiplying two square matrices in $\mathcal{M}_{n\times n}$ requires n^3 multiplications and $n^2(n-1)$ additions. In 1969, Strassen published a recursive algorithm that requires $n^{\log_2 7}$ multiplications for $\mathcal{M}_{n\times n}$ matrices provided that n is a power of 2 [10]. Since then, considerable effort has been

spent on finding algorithms that require fewer multiplications than the traditional algorithm at the price of more additions.

For matrices of size 3×3 it has been shown in [3] that $\sigma = 19$ is the lower bound for the number of multiplications required. The best known exact algorithms use 23 multiplications [7, 5], while the best known approximate algorithm uses 22 [1]. A recursive MM algorithm over $\mathcal{M}_{n\times n}$, where n is a power of 3, more efficient than Strassen's cannot use more than $n^{\log_3 21}$ multiplications (as $\log_3 21 < \log_2 7 < \log_3 22$).

In order to apply genetic algorithms (GA) to the discovery of MM algorithms, we need to define the search space, the fitness function and the genetic operators. Many of the concepts used in this investigation are based on [6].

Problem definition

A bilinear MM algorithm calculates the product $\boldsymbol{Z}$ of two matrices $\boldsymbol{X}$ and $\boldsymbol{Y} \in \mathcal{M}_{\delta\times\delta}$ using σ multiplications as

$$z_{mn} = \sum_{p=1}^{\delta} x_{mp} y_{pn} = \sum_{r=1}^{\sigma} \left(\sum_{i,j=1}^{\delta} a_{ij}^{r} x_{ij} \right) \left(\sum_{k,l=1}^{\delta} b_{kl}^{r} y_{kl} \right) c_{mn}^{r}. \tag{1}$$

The *search space* of bilinear MM algorithms is then determined by coefficient matrices $\boldsymbol{A^r}$,$\boldsymbol{B^r}$,$\boldsymbol{C^r} \in \mathcal{M}_{\delta\times\delta}, 1 \leq r \leq \sigma$. As noted in [6], $\boldsymbol{C^r}$ can be calculated based on $\boldsymbol{A^r}$ and $\boldsymbol{B^r}$, therefore $\boldsymbol{A^r}$ and $\boldsymbol{B^r}$ uniquely determine the MM algorithm.

GA search space

A genome $\boldsymbol{G} \equiv (\boldsymbol{A^r}, \boldsymbol{B^r})$ is defined as an element of $\mathbb{K}^{2\delta^2\sigma}$ where $\mathbb{K}$ is an arbitrary field. Table 1 shows the size of the genome defined by the number of elements making up a solution, the size of the search space considering all elements for

$\mathbb{K} = \{-1, 0, 1\}$ and the size of neighbourhood defined as the number of genomes which differ from a given genome in one position. The numbers indicate that enumeration of the whole space is not possible.

Table 1: Search space description.

δ	σ	Genome size $[G = 2\delta^2\sigma]$	Search space size $[S = 3^G]$	Neighbourhood size $[N = 2G]$
2	7	56	5.23e+26	112
3	21	378	2.25e+180	756
3	22	396	8.71e+188	792
3	23	414	3.37e+197	828

Fitness measure

As described in [6], for a genome $\boldsymbol{G}$ let $\boldsymbol{A} \equiv \boldsymbol{A}(\boldsymbol{G}) \in \mathcal{M}_{\delta^4 \times \sigma}$ the matrix that has as columns the Kronecker products $(\rho(\boldsymbol{A^r}) \otimes \rho(\boldsymbol{B^r}))^T$, where $\rho(\cdot)$ is an operator which rewrites its argument in row-major format. Consider the matrix $\boldsymbol{S} \in \mathcal{M}_{\delta^4 \times \delta^2}$ with elements $S_{\alpha\beta} = 1$ if $\alpha = \gamma(\delta^2 - 1) + \kappa$, $\beta = i(\delta - 1) + j$, $\kappa = k(\delta - 1) + j$, $\gamma = i(\delta - 1) + k$ and $1 \leq i, j, k \leq \delta$ and 0 otherwise.

The *multiplication error* of $\boldsymbol{G}$ is defined as $E(\boldsymbol{A}) = \boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{S} - \boldsymbol{S}$ if $\boldsymbol{A}^T\boldsymbol{A}$ is nonsingular and ∞ otherwise [6]. We define the *fitness* of $\boldsymbol{G}$ as $f(\boldsymbol{G}) = \dfrac{1}{1 + ||E(\boldsymbol{A})||}$, where $||E(\boldsymbol{A})|| = \mathrm{tr}(E(\boldsymbol{A})^T E(\boldsymbol{A}))$. We use this fitness function as (1) it is based on the error, (2) has a value of 1 corresponding to a perfect solution, (3) it has previously been used successfully for matrices of size 2 and (4) allows comparison with [6].

Representation

The representation of the genome consists of a fixed length string containing elements $a_{ij}^r, b_{kl}^r,\ 1 \leq i,j,k,l \leq \delta, 1 \leq r \leq \sigma$. GAs with ternary $\{-1,0,1\}$ encoding have led to MM algorithms equivalent to Strassen's algorithm [6, 8]. So far no GA has been shown to be capable of evolving MM algorithms for matrices of size 3×3.

We extended both methods [6, 8] to matrices in $\mathcal{M}_{3\times 3}$. Despite the parallel implementation no solution could be located. We consider the discrete (ternary) representation one of the main limitations of these approaches. Therefore, we propose to use a *continuous real-valued representation.*

Consider a nonsingular matrix $\boldsymbol{A} \in \mathcal{M}_{\delta^4\times\sigma}$ and matrices $\boldsymbol{A_1} \in \mathcal{M}_{\delta^4\times q}$ and $\boldsymbol{A_2} \in \mathcal{M}_{q\times\sigma}$, such that $\boldsymbol{A} = \boldsymbol{A_1}\boldsymbol{A_2}$. Then it is possible to prove that $E(\boldsymbol{A}) = E(\boldsymbol{A_1})$, and so, in particular, $E(\alpha\boldsymbol{A}) = E(\boldsymbol{A})$, $\alpha \in \mathbb{R}$. This means that using real genes of fixed precision of p decimals is equivalent to using integers, as $\boldsymbol{A}$ can be multiplied by $\alpha = 10^p$ to contain integer values only.

Genetic algorithm and search operators

We deployed a parallel GA for the evolution of MM algorithms for $\delta = 3$. We report the parameter settings that produced the best results. 64 islands run the same steady-state, elitist GA, evolving $20 \ldots 300$ individuals independently. Each individual is initialized based on allele values in the range [-50,50]. Fitness-proportionate selection, randomly selected crossover and mutation operators are applied with probabilities 0.9 and 0.05, respectively. The following crossover types were used:

1. Gaussian blend crossover. Each gene g_i in the offspring genome is calculated

using the corresponding parent genes g_i^f and g_i^m by sampling the normal distribution $\mathcal{N}(0.5(g_i^f + g_i^m), 0.5|g_i^f - g_i^m|)$.

2. Real blend crossover. Similar to the Gaussian blend crossover with the difference that offspring gene g_i is obtained by sampling uniformly $[g_i^f, g_i^m]$.

3. Quasi momentum crossover. First a difference vector d of the parents g^f and g^m is calculated as $d = \mathrm{sgn}\left(f(g^f) - f(g^m)\right)\left(g^f - g^m\right)$. The offspring's genes are calculated as $g_i^1 = g_i^m + \gamma d_i$ and $g_i^2 = g_i^f + \gamma d_i$, where is γ is a granularity parameter generated uniformly from $[0.1, 0.5]$.

The following types of mutation were employed:

1. Gaussian mutation. The selected gene g_i is mutated according to $g_i = g_i + \mu\mathcal{N}(0, 1)$, where μ denotes the mutation step. Adaptivity has been built into Gaussian mutation as follows. Let f_1, f_2 and f_3 be the fitness values of the best individuals measured at consecutive equal time intervals. The mutation step μ is updated according to the relative amount of improvement observed, i.e. $\mu = (1 - 0.1\alpha)\mu$, where $\alpha = \mathrm{sgn}(2f_2 - f_1 - f_3)$.

2. Simple mutation. The selected gene g_i is mutated according to $g_i = g_i + \kappa$, where κ is sampled uniformly from $[-10, 10]$.

3. Permutation mutation. Two randomly selected genes are swapped.

As GAs are prone to premature convergence, we *explicitly enforced diversity* by disallowing individuals with *the same genotype or fitness* in a population.

The islands are organized in a standard unidirectional ring topology [4]. At every 5000 generations the best individual together with $5 \ldots 50$ randomly selected individuals from each island migrate according to the ring topology.

The termination condition was the discovery of a solution of fitness $1 - 10^{-10}$ or by manual termination.

The genetic algorithm was implemented in C++ based on the BOOST, GALib and Armadillo [9] libraries. The hardware platform consisted of a SGI cluster featuring 32 nodes 8-core Intel Xeon 2.5 GHz processors.

Results

The convergence of the best fitness on all individual islands using 23 and 22 multiplications are depicted in Figure 1. For 23 multiplications a perfect solution is reached. For 22 multiplications a solution of fitness 0.9978 is found[1]. For 21 multiplications the best fitness was below 0.5.

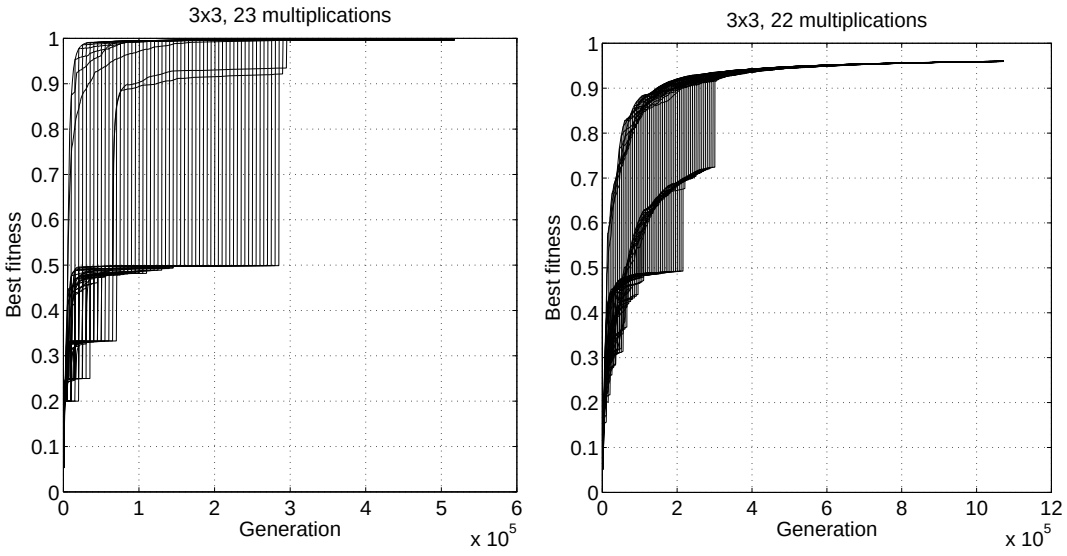

Figure 1: GA convergence for 23 and 22 multiplications. The vertical jumps in fitness are due to the migration of the best individuals.

We compared the quality of our GA to its predecessors, Kolen and Bruce's algorithm (KB) [6] and Oh and Moon's algorithm (OM) [8].

[1]The best individual is available in the supplementary material on `http://ieeexplore.ieee.org`.

We implemented a parallel version of KB. We tried to keep the parameters and features as described in the article. Population size 300, cross-breeding, uniform crossover, mutation with probability 0.7 and introduction of 10 random individuals in each generation were implemented. Similarly to [6] fitness penalties were applied to sparse $\boldsymbol{A}$ matrices and to aging individuals. KB was parallelised for ring topology with 64 islands, migration rate of 10 and migration frequency of 10. The algorithm was allowed to run for 4.2×10^5 generations. The evolution of fitness for 23 and 22 multiplications is shown in Figure 2. The best fitness could not reach a value above 0.11 in any of the runs.

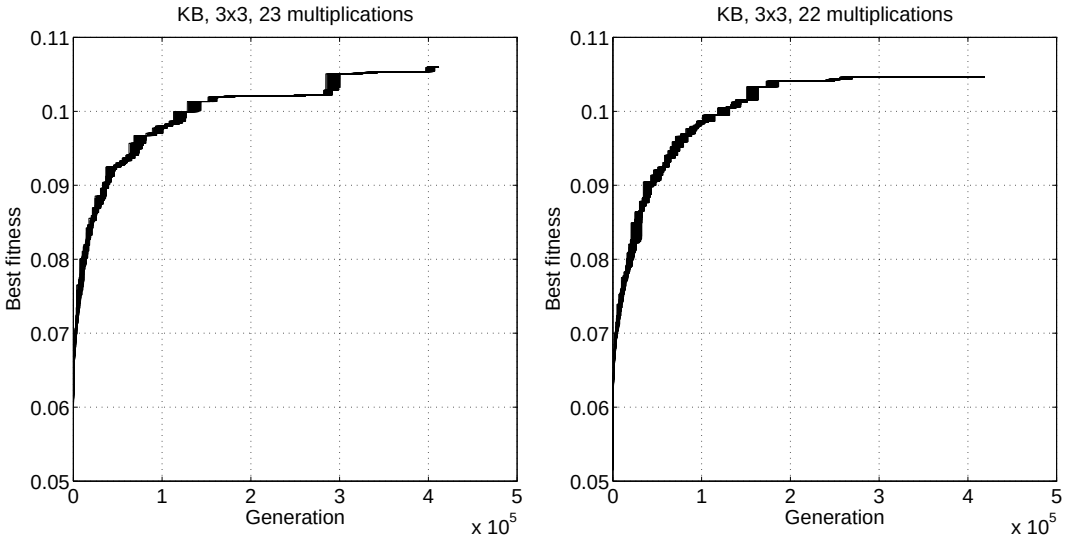

Figure 2: KB convergence for 23 and 22 multiplications.

We also implemented a parallelised version of OM scaled for 3×3 matrices. The implementation cannot follow the OM algorithm in all respects due to the mutation including an exhaustive local search (ELS). For 2×2 matrices, ELS has to consider 1600 cases for each individual. ELS is technically not feasible for square matrices of size 3, as it would need to consider 9.684×10^7 cases. Therefore we replaced ELS with the checking of 1600 random cases. Put in the same parallel

framework as our GA, OM could not find any individual of non-zero fitness.

We employ an evaluation method for assessing the quality of approximate MM methods described by Bard [2]. We define the multiplication error of a MM algorithm Λ on $\mathbf{X},\mathbf{Y} \in \mathcal{M}_{\delta\times\delta}$ as $\varepsilon(f|\mathbf{X},\mathbf{Y}) \equiv \|f(\mathbf{X},\mathbf{Y}) - \mathbf{XY}\|_\infty$, where $\|\mathbf{X}\|_\infty = \max_i \sum_j |\mathbf{X}_{ij}|$. In the characterization of a MM algorithm Λ we calculated the mean and standard deviation of ε averaged over the 81 pairs $\mathbf{B}^i$, $\mathbf{B}^j$ of elements from the canonical basis of $\mathcal{M}_{\delta\times\delta}[\mathbb{R}]$. We found that $\varepsilon = 0.25\pm0.13$ for the parallel KB whilst $\varepsilon = 0.0055 \pm 0.0031$ for our genetic algorithm.

Conclusion

We have presented an application of parallel GAs to find the parameters of MM algorithms for square matrices. The required number of multiplications is a parameter of our algorithm, so various values within the range from the largest known lower bound to the currently best known number can be used in the hope of finding better algorithms (i.e. with fewer multiplications) than the currently known best ones.

Our algorithm successfully discovered solutions requiring 23 multiplications for square matrices of size 3, which are equivalent to the currently best known human-developed algorithms. For 22 multiplications, our algorithm found an approximate solution of fitness 0.9978. We believe that the key to finding a perfect solution, if there exists one, is fitness landscape analysis followed by the design of genetic operators according to this analysis. In summary, in our opinion the success of our approach has four major components: 1) The use of continuous representation instead of discrete representation, 2) the use of a parallel strategy, 3) the use of migration and explicit diversity maintenance to avoid premature convergence and

4) the use of a variety of genetic operators.

References

[1] G. Bard. New practical Strassen-like approximate matrix multiplication algorithms found via solving a system of cubic equations. `http://www-users.math.umd.edu/~bardg/publications.html`.

[2] G. Bard. A practical algorithm for massively-parallel dense matrix multiplication in time $n^{2.777}$ over any field someday, but for now, the reals, 2008. `http://www.usna.edu/Users/math/wdj/colloq/bard-usna-talk2008.pdf`.

[3] M. Bläser. Lower bounds for the multiplicative complexity of matrix multiplication. *Computational Complexity*, 8:203–226, 1999.

[4] E. Cantú-Paz. Migration policies, selection pressure, and parallel evolutionary algorithms. *Journal of Heuristics*, 7(4):311–334, 2001.

[5] R. W. Johnson and A. M. McLoughlin. Noncommutative bilinear algorithms for 3×3 matrix multiplication. *SIAM J. Comput.*, 2:595–603, 1986.

[6] J. F. Kolen and P. Bruce. Evolutionary search for matrix multiplication algorithms. In *FLAIRS*, pages 161–165, 2001.

[7] J. D. Laderman. A noncommutative algorithm for multiplying 3 x 3 matrices using 23 multiplications. *Bulletin of the American Mathematical Society*, 82(1):126–128, 1976.

[8] S. Oh and B.-R. Moon. Automatic reproduction of a genius algorithm: Strassen's algorithm revisited by genetic search. *IEEE Transactions on Evolutionary Computation*, 14(2):246–251, April 2010.

[9] C. Sanderson. Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments. Technical report, NICTA, 2010.

[10] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.