SPEED OPTIMIZATION OF MATLAB CODES

Hristo Zhivomirov, Todor Vachev, Iva Todorova

Abstract: In the present paper algorithmic and software approaches are discussed, providing speed optimization of the computational process in the execution of Matlab® codes. A major consideration is made on the loop structures, which are an essential source of delay. The statement is supported by examples that give a concrete idea of the used optimization approaches. At the end of the publication the results of the examples execution are summarized by means of tabular and graphic representation of the absolute and relative codes speed up.
Keywords: Matlab, optimization, program code, speed

INTRODUCTION

The modern ways of information and signal processing by means of computers and other microprocessor systems, require that computing process is held in real time. For this reason the computational speed of these systems is one of their most important parameters.

There are two main approaches for enhancing the execution of program codes. The first is hardware improvement, evidence for which is the constant increase of the clock speed of microprocessor systems and an increase in the processed bits per clock. The second approach is to optimize the code itself, by using enhanced algorithms or the specific advantages of the given programming environment. In the present paper the second approach is taken in consideration with Matlab® implementation. Some of the reviewed methods are general to all high-level programming languages, others are specific for Matlab®.

The main source of delay is assumed to be a CPU performance, i.e. the codes is said to be a “CPU-bounded”, so more of the described strategies is in this direction. A good reference about the “memory-bounded” codes is [1].

METHODS FOR OPTIMIZATION

Six optimization methods have been reviewed, each one of them supported by two examples – those with index “a)” show the non-optimized Matlab® code, and those with index “b)” show the optimized one.

1. Memory Preallocation

The first and most commonly used method for optimization, producing significant results is memory preoallocation [2, 3, 4]. The avoiding of fragmentation, especially with big arrays of data, leads to a quick access to them, which enhances the execution of the program code. In the Matlab® programming environment there are two commands used for memory preallocation: zeros – to create a matrix filled with zeros and char – to create a string array filled with symbols.

Example 1a:
```
clear, clc, close all
for k = 1:1000
   a(k) = k^2;
end
```

Example 1b:
```
clear, clc, close all
a = zeros(1, 1000);
for k = 1:1000
   a(k) = k^2;
end
```

2. Vectorization

Significant attention in Matlab® is given to matrices and their one-dimensional version – vectors. The name of the programming environment itself means Matrix Laboratory [5]. Matlab® offers easy-to-use method for vector generation [4, 5]:
```
vector = 1:1:1000;
```

or the linspace function:
vector = linspace(1, 1000, 1000);

The generation of vectors (vectorization) and matrices with special commands [2, 3, 4], instead of the most commonly used loop-structures results in a significant enhancement of the execution process of the code.

Example 2a:
```
clear, clc, close all
k = 0;
for t = 0:pi/1000:2*pi
   k = k + 1;
   y(k) = sin(t);
end
```

Example 2b:
```
clear, clc, close all
t = 0:pi/1000:2*pi;
y = sin(t);
```

3. Loops exchange

Another way to enhance the execution process is a simple exchange of nested loops [6].

Example 3a:
```
1  clear, clc, close all
2
3  a = [];
4
5  for n = 1:1000
6
7     for k = 1:5
8        a(n, k) = n*k;
9     end
10
11 end
```

Example 3b:
```
1  clear, clc, close all
2
3  a = [];
4
5  for n = 1:5
6
7     for k = 1:1000
8        a(k, n) = n*k;
9     end
10
11 end
```

Here the optimization is achieved thanks to the difference in the number of iterations of some code lines as is shown in Tab. 1. As a rule of thumb, the outer loop should have less iterations than the inner one.

Table 1. Number of iterations (executions) of code lines in examples 3a) and 3b)

Line	Number of iterations	
	Example 3a)	Example 3b)
1	1	1
3	1	1
5	1	1
7	1000	5
8	5000	5000
9	5000	5000
11	1000	5
Total iterations:	12003	10013

4. Extract a constant out of a loop

Another approach is based on the extracting of specific operations outside the body of the loop. This approach does not confine to nested loops (as is shown in the example) but holds true also for single loops. The diversity of the method is the algorithmic optimization, which is based on the full revision of the used algorithm, especially if there are conditional statements and loop structures [6].

Example 4a:
```
clear, clc, close all
for k = 1:1000
   for n = 1:1000
      a(k, n) = k*5+n;
   end
end
```

Example 4b:
```
clear, clc, close all
for k = 1:1000
   b = k*5;
   for n = 1:1000
      a(k, n) = b+n;
   end
 end
```

5. Exchange of computational operations

It is known that in the computational processes the different mathematical operations

have different priorities [6, 7]. The less priority the operation has, the more time it needs to be executed [8]. Therefore it is preferable to use operations with a higher priority.

```
Example 5a:
clear, clc, close all
for k = 1:1000
   a(k) = k/10;
end

Example 5b:
clear, clc, close all
for k = 1:1000
   a(k) = k*0.1;
end
```

6. Assign a constant into a variable

Additional enhancement of the execution of the program code can be achieved by exchanging all constants with variables [6].

```
Example 6a:
clear, clc, close all
for k = 1:1000
   a(k) = k*1.4142;
end

Example 6b:
clear, clc, close all
c = 1.4142;
 for k = 1:1000
   a(k) = k*c;
end
```

Other techniques for increasing the performance of the Matlab® codes are [3]:

- splitting the large Matlab® scripts into smaller ones in the form of file-functions since they are generally faster;
- avoiding the change of the class or the size of an existing variables since it takes extra time to process;
- using appropriate logical operators like “short-circuit” OR (||) and AND (&&);
- avoiding the execution of large processes in the background while running a program in Matlab®.

One must be aware of the effect of some techniques since they can affect the amount of used memory or the speed of computation if the code is “memory bounded” [1].

CONCLUSION

The examples have been tested on three different computer configurations and the execution time of every test is reported by the Matlab® commands tic and toc [2, 3] and the Matlab® Profiler [9]. The specifications of the configurations are shown in Table 2. The results from the tests are shown in Tables 3, 4, 5 and Fig. 1.

Due to the differences in the execution time for every example, the tests were performed 5 consecutive times, and the average time for every example is taken. The time of execution of a given code depends on the computer configuration (mainly by the clock speed of the processor). Despite that the coefficient of relative speed-up (in %) gives an opportunity to objectively compare the results.

One can note that the result of example 4 with the first configuration is due to the slow access of the DDR1 RAM memory and 32-bit operation system. Also it must be noted the big difference (ten times) between the execution times of Example 1a) with configuration 1 and 3. This is an evidence for the effect of the hardware acceleration on the speed of computation.

Table 2. Performance specifications of the used computer configurations

Configuration	1	2	3
OS	Windows XP 32 bit	Windows 7 64 bit	Windows 7 64 bit
Processor	AMD Athlon 64 Processor 3000+ 1,81 GHz	AMD Turion II P560 Dual-Core Processor 2,50 GHz	AMD FX-8150 Bulldozer Eight-Core Processor 4,60 GHz
RAM	DDR 1 GB	DDR3 4 GB	DDR3 16 GB
Matlab	R2010b	R2010b	R2010b

Table 3. Time parameters from tests with the shown examples on configuration 1

№	$t_{non-opt}$, ms	t_{opt}, ms	$\Delta t = t_{non-opt} - t_{opt}$, ms	$\frac{\Delta t}{t_{non-opt}} \cdot 100$, %
1	4,5	0,026	4,47	99,4
2	27	0,2	26,8	99,2
3	13,3	3,2	10,1	75,9
4	9010	8881	129	1,43
5	4,1	3,7	0,4	9,7
6	5,1	5	0.1	1,9

Table 4. Time parameters from tests with the shown examples on configuration 2

№	$t_{non-opt}$, ms	t_{opt}, ms	$\Delta t = t_{non-opt} - t_{opt}$, ms	$\frac{\Delta t}{t_{non-opt}} \cdot 100$, %
1	4,2	0,05	4,15	98,8
2	24	0,37	23,63	98,5
3	11,9	4,4	7,5	63
4	2084	1948	136	6,5
5	4,5	4,1	0,4	8,8
6	3,9	3,8	0,1	2,5

Table 5. Time parameters from tests with the shown examples on configuration 3

№	$t_{non-opt}$, ms	t_{opt}, ms	$\Delta t = t_{non-opt} - t_{opt}$, ms	$\frac{\Delta t}{t_{non-opt}} \cdot 100$, %
1	0,45	0,015	0,435	96,67
2	0,95	0,062	0,888	93,47
3	1,85	0,49	1,36	73,51
4	472	406	66	13,98
5	0,43	0,39	0,04	9,3
6	0,42	0,37	0,05	11,91

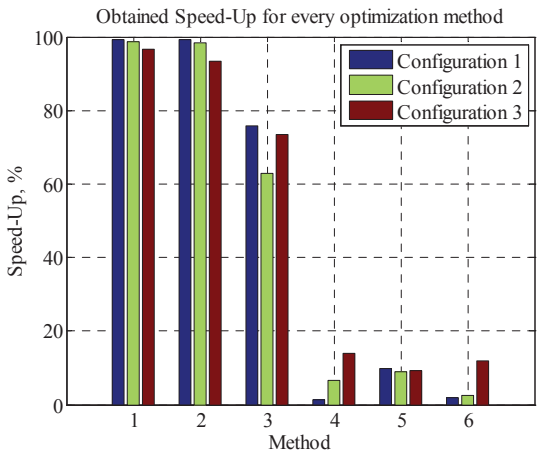

Fig. 1. Graphical representation of obtained speed-up by every optimization method

In conclusion one can say that by using those easily applicable methods a serious enhancement of the execution speed of the codes can be managed. Many of the methods can be used by other programming environments/languages.

REFERENCE

[1] S. McGarrity. Maximizing Code Performance by Optimizing Memory Access. The MathWorks News & Notes, June 2007.
[2] P. Getreuer. Writing Fast Matlab Code, http://www.getreuer.info/tutorials, 2009.
[3] Techniques for improving performance, http://www.mathworks.com/help/matlab/matlab_prog/techniques-for-improving-performance.html (Accessed Dec. 2014)
[4] Vectorization, http://www.mathworks.com/help/matlab/matlab_prog/vectorization.html (Accessed Dec. 2014)
[5] J. Tonchev. Matlab 7: part I (in Bulgarian). Sofia, Technika, 2007.
[6] A. Angelov. The Programming – simple and complex (in Bulgarian). Sofia, Technika, 1986.
[7] H. Warren. Hacker’s Delight. Boston, Addison-Wesley, 2012.
[8] S. Oliveira, D. Stewart. Writing Scientific Software: A Guide to Good Style. Cambridge, Cambridge University Press, 2006.
[9] Profiling for Improving Performance, http://www.mathworks.com/help/matlab/matlab_prog/profiling-for-improving-performance.html (Accessed Dec. 2014)

Contacts:
Assist. Prof. M.Sc. Eng. Hristo Zhivomirov,
Dept. of Theory of Electrical Engineering and Measurements,
Technical University-Varna,
Str. Studentska 1,
e-mail: hristo_car@abv.bg;