Accelerating MATLAB Algorithms and Applications

By Sarah Wait Zaranek, Bill Chou, Gaurav Sharma, and Houman Zarrinkoub, MathWorks

This article describes techniques that you can use to accelerate your MATLAB® algorithms and applications. Topics covered include:

- Assessing code performance
- Adopting efficient serial programming practices
- Working with System objects
- Performing parallel computing on multicore processors and GPUs
- Generating C code

Each section focuses on a particular technique, describes the underlying acceleration technology, and explains when it is most applicable. Your programming expertise, the type of algorithms you wish to accelerate, and the hardware you have available can help guide your selection of techniques.

Assessing Code Performance

Before modifying your code, you need to determine where to focus your efforts. Two critical tools to support this process are the Code Analyzer and the MATLAB Profiler. The Code Analyzer in the MATLAB Editor checks your code while you are writing it. The Code Analyzer identifies potential problems and recommends modifications to maximize performance and maintainability (Figure 1). Code Analyzer Reports can be run on entire folders, enabling you to view all Code Analyzer recommendations for a set of files in a single document.

The Profiler shows where your code is spending its time. It provides a report summarizing the execution of your code, including the list of all functions called, the number of times each function was called, and the total time spent within each function (Figure 2). The Profiler also provides timing information about each function, such as which lines of code use the most processing time.
Once you have identified the bottlenecks, you can focus on ways to improve the performance of those particular sections of code. As you implement optimizations and techniques to speed up your algorithm, the Profiler can help you measure the improvement.

**Adopting Efficient Serial Programming Practices**

It is often a good practice to optimize your serial code for performance before considering parallel computing, code generation, or other approaches. Two effective programming techniques to accelerate your MATLAB code are **preallocation** and **vectorization**.

With preallocation, you initialize an array using the final size required for that array. Preallocation helps you avoid dynamically resizing arrays, particularly when code contains `for` and `while` loops. Since arrays in MATLAB are held in contiguous blocks of memory, repeatedly resizing arrays often requires MATLAB to spend time looking for larger contiguous blocks of memory and then moving the array into those blocks. By preallocating arrays, you can avoid these unnecessary memory operations and improve overall execution time.

Vectorization is the process of converting code from using loops to using matrix and vector operations. MATLAB uses processor-optimized libraries for matrix and vector computations. As a result, you can often improve performance by vectorizing your code.

Vectorized MATLAB calculations that use larger arrays may be good candidates for acceleration using a GPU. In cases where `for`-loops cannot be vectorized, you can often use a parallel `for`-loop (`parfor`) or C-code generation to accelerate the algorithm. See the sections on parallel computing and generating C code from MATLAB for more details on these techniques.

Learn more about **profiling**, **preallocation**, **vectorization**, and *other serial techniques* to improve performance.

**Leveraging Optimized Algorithms Using System Objects**

You can use **System objects** to accelerate MATLAB code largely in the areas of signal processing and communications. System objects are MATLAB object-oriented implementations of algorithms available in System toolboxes, including Communications System Toolbox and DSP System Toolbox. By using System objects, you decouple declaration (System object creation) from the execution of the algorithms found in the System object. This decoupling results in more efficient loop-based calculations, since it lets you perform the parameter handling and initializations only once. You can create and configure an instance of a System object outside the loop and then call the step method inside the loop to execute it.

Most System objects in DSP System Toolbox and Communications System Toolbox are implemented as MATLAB executables (MEX-files). This implementation can speed up simulation, since many algorithmic optimizations are included in the MEX implementations of the objects. See the section on generating C code from MATLAB for more details on MEX-files.
Learn more about stream processing with System objects, creating your own System objects, and simulating signal processing algorithms in MATLAB using System objects.

**Performing Parallel Computing**

The techniques described so far have focused on ways to optimize serial MATLAB code. You can also gain performance improvements by using additional computing power. MATLAB parallel computing products provide computing techniques that let you take advantage of multicore processors, computer clusters, and GPUs.

**Using MATLAB Workers on Multicore Processors and Clusters**

Parallel Computing Toolbox™ lets you run multiple MATLAB workers (MATLAB computational engines) on your desktop multicore machine. You can speed up your applications by dividing computations across these workers. This approach gives you more control over the parallelism than with the built-in multithreading found in MATLAB. It is often used for coarser-grained problems such as parameter sweeps and Monte Carlo simulations. For greater speedup, parallel applications that use MATLAB workers can be scaled to a computer cluster or cloud using MATLAB Distributed Computing Server™.

Several toolboxes, including Optimization Toolbox™ and Statistics Toolbox™, provide algorithms that can utilize multiworker parallelism to accelerate your computations. In most cases, you can use the parallel algorithms by simply turning on an option. For example, to run `fmincon` in Optimization Toolbox in parallel you set the 'UseParallel' option to 'always'.

Parallel Computing Toolbox offers high-level programing constructs such as `parfor`. Using `parfor` you can accelerate `for`-loops in your MATLAB code by dividing loop iterations for simultaneous execution across several MATLAB workers (Figure 3).

![Figure 3. Iterations of a `parfor`-loop distributed across multiple MATLAB workers on a multicore computer.](image)

To use `parfor`, the loop iterations must be independent, with no iteration dependent on any other. To accelerate dependent or state-based loops, you can reorder computations so that the loop becomes order-independent. Alternatively, you can parallelize an outer loop that contains the `for`-loop. If these options are not feasible, either optimize the body of the `for`-loop or consider generating C code instead.

By transferring data between the client and MATLAB workers for `parfor` loops, you incur a communication cost. This means that there might be no advantage to using `parfor` when you have only a small number of simple calculations. If that is the case, focus instead on parallelizing an outer `for`-loop that contains the simpler `for`-loop.
The `batch` command can be used for distributing independent sets of computations across MATLAB workers for offline processing as batch jobs. This approach is particularly useful when these computations take a long time to run and you need to free up your desktop MATLAB for other work.

**Using GPUs**

Originally used to accelerate graphics rendering, GPUs (graphics processing units) can also be applied to scientific calculations in signal processing, computational finance, energy production, and other areas.

You can perform computations on NVIDIA GPUs directly from MATLAB. FFT, IFFT, and linear algebraic operations are among more than 100 built-in MATLAB functions that can be executed directly on the GPU. These overloaded functions operate on either the GPU or the CPU, depending on the data type of the arguments passed to them. When given an input argument of a GPUArray (a special array type provided by Parallel Computing Toolbox) these functions will automatically run on the GPU (Figure 4). Several toolboxes, including Communications System Toolbox and Signal Processing Toolbox, also provide GPU-accelerated algorithms.

![Serial code (left) and code converted to run on a GPU (right) using overloaded functions for GPUArrays.](#)

Two rules of thumb will ensure that your computationally intensive problem is a good fit for the GPU. First, you will see the best performance on the GPU when all the cores are kept busy, exploiting the inherently parallel nature of the GPU. Code that uses vectorized MATLAB calculations on larger arrays and the GPU-enabled toolbox functions fits into this category. Second, the time required for the application to run on the GPU should be significantly more than the time required to transfer data between CPU and GPU during the application execution.

For more advanced use of GPUs, if you are familiar with CUDA programming, you can run existing CUDA-based GPU kernels directly from MATLAB. You can then use the data analysis and visualization capabilities in MATLAB while having more direct control over your GPU algorithm.

Learn more about using `parfor` and `batch`, running MATLAB on multicore and multiprocessor machines, GPU computing with MATLAB, and toolboxes with built-in parallel and GPU-enabled algorithms.

**Generating C Code from MATLAB Code**

Replacing parts of your MATLAB code with an automatically generated MATLAB executable (MEX-function) may yield speedups. Using MATLAB Coder, you can generate readable and portable C code and compile it into a MEX-function that replaces the equivalent...
section of your MATLAB algorithm (Figure 5). You can also take advantage of multicore processors by generating MEX-functions from `parfor` constructs.

![MATLAB Coder menu for generating a MEX-function.](image)

**Figure 5. MATLAB Coder menu for generating a MEX-function.**

The amount of acceleration achieved depends on the nature of the algorithm. The best way to determine the acceleration is to generate a MEX-function using MATLAB Coder and test the speedup first hand. If your algorithm contains single-precision data types, fixed-point data types, loops with states, or code that cannot be vectorized, you are likely to see speedups. On the other hand, if your algorithm contains MATLAB implicitly multithreaded computations such as `fft` and `svd`, functions that call IPP or BLAS libraries, functions optimized for execution in MATLAB on a PC such as FFTs, or algorithms where you can vectorize the code, speedups are less likely. Try MATLAB Coder, follow best practices for C-code generation, and consult MathWorks technical experts to find the best methods for accelerating your algorithm with this approach.

Much of the MATLAB language and several toolboxes support code generation. MATLAB Coder provides automated tools to help you assess the code-generation readiness of your algorithm and guide you through the steps to C-code generation (Figure 6).

![MATLAB tool for assessing code-generation readiness and guide you to start generating C code.](image)

**Figure 6. MATLAB tool for assessing code-generation readiness and guide you to start generating C code.**

Learn more about getting from MATLAB to C code and how to quickly get started with MATLAB Coder.
**Possible Performance Gains**

You can accelerate your MATLAB applications through writing efficient algorithms, parallel processing, and code generation. Each method has a range of possible speedups, depending on the problem and the hardware you are using. The benchmarks and acceleration examples listed here give a general idea of the accelerations that are possible.

Learn more about performance gains using `parfor`, different types of supported GPU functionality, built-in GPU support for System objects, and C-code generation.

**Combining Techniques**

You can often achieve additional acceleration by combining the methods described in this article. For example, `parfor`-loops can call C-based MEX-functions, code generation is supported for many System objects, and vectorized MATLAB code can be adapted to run on a GPU.

Learn more about using multiple techniques together to accelerate the simulation of communications algorithms and the design of a 4G LTE system.

---

1. This article does not cover performance limitations caused by memory issues such as swapping or file I/O. For more information on these topics, see strategies for efficient use of memory and data import and export.

2. When used with Parallel Computing Toolbox.

---

**Products Used**

- MATLAB
- Communications System Toolbox
- DSP System Toolbox
- MATLAB Coder
- MATLAB Distributed Computing Server
- Optimization Toolbox
- Parallel Computing Toolbox
- Signal Processing Toolbox
- Statistics Toolbox

**Learn More**

- Speeding Up MATLAB Applications
- Parallel Computing with MATLAB on Multicore Desktops and GPUs
- MATLAB to C Made Easy

---

See more articles and subscribe at mathworks.com/newsletters.