

MULTITHREADING

APPLICATION IN MODERN APPLICATIONS

NIKOLA PREMCEVSKI
FEBRUARY 2ND, 2016

TABLE OF CONTENTS

INTRODUCTION.....	3
BACKGROUND.....	3
PROBLEMS.....	3
ADVANTAGES.....	3
APPLICATIONS.....	4
REAL-LIFE EXAMPLES.....	4
IMAGES.....	4

INTRODUCTION

BACKGROUND

Multithreading is a way of making computers do multiple things at once, but compared to technics like time-slicing and similar, multithreading allows actual execution of multiple operations at the same time. This is possible due to existence of multiple processor cores, or even processors inside one machine. Multicore systems were introduced in the early 21st century, yet still not many applications allow true usage of even two, and especially not three or more cores.

PROBLEMS

Being able to do more things at once requires either working with separate resources, or technics of preventing so-called collisions between threads upon accessing the same resource. Both make implementing multithreading into existing applications difficult, and the programmers often ask themselves if it's worth it. Not always can you make more things at once, depending on the type of data you're working with, and how and what you're working with it. Although still not very widely used, it is a good thing to try and implement multithreading whenever possible.

ADVANTAGES

Main advantage of using multithreading is linear performance gain in term of execution time per every processor core dedicated to the task – twice the cores, double the execution speed, and half the execution time. Of course, although we say each core acts as a separate processing unit and that each core increases the performance by 100%, it is not exactly true, as some resources are still shared across the cores, like the registries, and the other resources of the computer, like the main memory, the hard-drive etc. Only thing that increases with each core is the computational power of arithmetical and logical operations. Because these two make up the most of every execution we usually say it's almost 100% faster per each core available.

APPLICATIONS

Multithreading can and should be applied where, and whenever we see a need to speed up the things a bit and there's not much to do regarding usual optimizations of code. Unless prevented by self-recursion or any other obvious problems which make parallel calculations impossible, parallelization could be possible, one way or the other.

Some things like array operations, matrix operations, or any type of calculation with little or no connections between separate “objects” just screams for multithreading. Array and matrix elements are stored inside their own cells and have no actual link among them, making it possible to manipulate them without having to worry about data corruption or similar. As they are stored in separate memory blocks, it is also possible to access them at the same time – as we know, computer's main memory (RAM) can be accessed at any time, and theoretically, all the memory can be accessed at the same time.

REAL-LIFE EXAMPLES

As we already said, arrays and matrices are a perfect candidate for parallelization, meaning if we can transform a real-life scenario into an array/matrix scenario, it can be improved using multithreading.

IMAGES

In digital computers, everything we see on the screen is actually composed of pixels, tiny dots on the screen represented with three color components – Red, Green and Blue (RGB). To improve the things a bit, a fourth component is also used in imagery, called transparency, or more commonly – Alpha (A). Therefore, images in computers are represented as a matrix of four components – RGBA.

All four components are actually a number which represents the amount of each component inside the current pixel. The number usually ranges from 0 to 255, making it a perfect candidate of type “byte”. This makes it easy to represent every pixel with 4 “byte” variables, sometimes simplified as a single 32 bit (4B) “int” variable – highest byte representing R component, and lowest – A component.

Now that we understand how images are stored inside the computer, manipulating an image with multiple threads is rather simple, we split the matrix into several sub-matrices and give each thread a single sub-matrix to work on, depending on the task we wish to do. As every pixel is a separate “object”, its value can be modified regardless of other pixels, meaning no collisions will happen during the manipulation.