STM for Games

Final Report

Author: Arkadiusz Bielecki
ID: C00139358
Date: 24.04.2014
Project Supervisor: Joseph Kehoe

Table of Contents

Introduction

This document summarizes all of the work done throughout every phase of developing STM for Games application. It describes every file contained in the project, and shows what its functions are. The summary of every aspect of building the STM for Games project and the final results and comparisons are also contained in the final report.

About My Project

The main purpose of my project is to show how Software Transactional Memory (STM) (a concurrency control mechanism) will affect the performance in gaming. I have implemented a spaceship game. It measures the performance using a standard synchronization, then using STM algorithms, benchmarks them, and compares whether the performance have increased or not. The performance will be measured under 1-, 2-, and 4-core processor configurations and measurement will be based on the screen refreshing rate (in frames per second - more fps means that the configuration runs faster and smoother). The application is written in C++ using Visual Studio. It contains Simple and Fast Multimedia Library to deal with the game objects, OpenMP algorithms for parallelizing the application on multiple CPU cores and Rochester Software Transactional Memory Library for implementing STM functions.

Main Objectives

The main objective of my project is to show how Software Transactional Memory (STM) (a concurrency control mechanism) will affect the performance in gaming.

All of the different configurations will consist of different configurations described below:

Algorithms used:

- **Swarming** - it was developed from observing the animal world. Its based on self-

organisation, interactions, feedback and fluctuations.

- **Flocking** - has been also taken from the animals interactions.
  Basic models of flocking behavior are controlled by three simple rules:
  1. Separation - avoid crowding neighbors (short range repulsion)
  2. Alignment - steer towards average heading of neighbors
  3. Cohesion - steer towards average position of neighbors (long range attraction)

Runs on:

- Multiple processor cores simultaneously

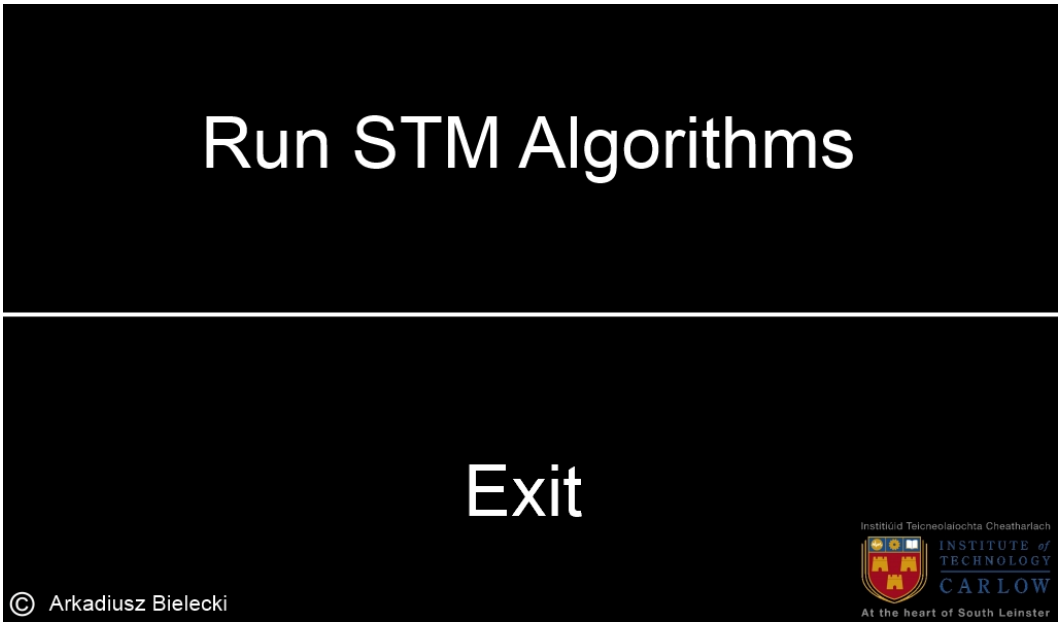
Available game window size:

- **Small** - board size of 640×480 pixels used to measure the output of performance.
- **Medium** - board size of 1280×800 pixels used (more calculations needed).
- **Large** - board size of 1600x1200 pixels used (even more calculations needed).

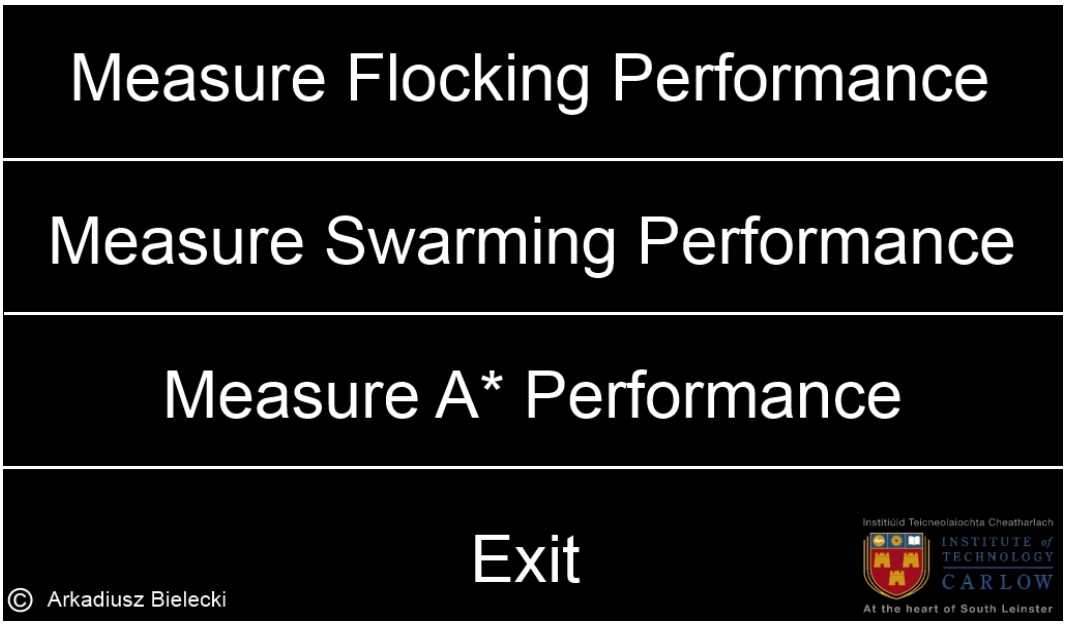
Frames Per Second will be measured and compared under various configurations.

Screenshots:

Project main menu:



Project algorithms menu:



Description of conformance to the specification and design

I had to use RSTM version 5 library instead of version 7, because version 5 is the latest released RSTM version that's compatible with Windows operating system.

Otherwise the code is compliant to the specification and design documents.

Description of learning

Technical Learning

Throughout the development of my project I've learned many things, as explained below:

- Using vectors in Visual Studio
    - calculating, adding and subtracting vectors
    - normalizing a vector

- How to implement and use Simple and Fast Multimedia Library
    - working with sprites from .png files
    - using boundary checks and collision detection
    - practice with various game objects and how they interact with each other

- Operation of various algorithms
    - what is a flocking and swarming, and how to implement them as an algorithms into programming

- Parallelization
    - how to add parallelization to the project
    - OpenMP library - implementation
    - spreading workload throughout multiple CPU cores, while keeping the program's flow of execution in order and full functionality

- Software Transactional Memory
    - what is and how to use RSTM (Rochester Software Transactional Memory) library

- building RSTM library and adding it to Microsoft Visual Studio
- various STM commands that can be used within the library

Personal Learning

The main things that I have learned while developing my project were:

- following the project according to the schedule (project plan)
- dealing with time management
- setting small targets for the project
- organising to increase productivity and efficiency
- dealing with stress management

Review of the project

Problems encountered and how they were resolved

During the development of my project I've run into various problems.

One of them was working with vectors of various game objects. Every object has to have its own vector (speed and direction associated with it). My problem had to do with the collision detection when objects were near each other, I had to reverse the vector direction accordingly when that happened. After learning how vectors work and doing some tweaks in my code, I was able to improve the physics of my game.

Another problem that I had was adding parallelization to my project. Importing OpemMP library itself is an easy part of the task, but when I did that my game objects started to move in random direction. I dealt with it by parallelizing inner for loop in a specific sequence, before parallelizing the outer for loop.

Adding RSTM to the Visual Studio was the toughest job of my whole project. I left it to the end which probably wasn't the best idea. After making my project in Visual Studio 2012 under Windows 7 operating system, I have realized that the latest version of RSTM (version 7) is not compatible with Windows. After struggle with building the library (RSTM need to be build under a specific configuration before it can be accessed) I have managed to add version 5, which is 6 years older than the newest one, into my project.

Last one of my problems was adding the STM commands into the code in my application. The instructions on the website weren't clear as to which commands to use in my version of RSTM. I am still working on adding it, because I haven't yet achieved it.

What is missing still to do

STM commands need to be added to the project in order to compare the algorithms.

What I would do differently if starting again

I wouldn't waste my time compiling RSTM version 7 into my project, as now I know that it can't be done.
If I would start working with the above library earlier on through the development of my project, I would have more time to make it work properly, and get the full functionality of my application.

What advice would I give for someone attempting a similar project

Make sure that you use the proper STM library accordingly to your system and technologies. Implement and try if its working first before going any further into the development.

Were technology choices right ones ?

Most of the technologies used in my project were spot on.

The big problem I have came into is that RSTM library is pretty much obsolete, it latest version came out in 2011, but the last Windows compatible version is version 5 from the year 2005. Microsoft stopped STM functionality in the 2009 version of Visual Studio, although I've been able to add it manually, it seems like some of the functionalities was lost. There really wasn't any other choice associated with Software Transactional Memory library for me to use.

Summary

Software Transactional Memory has become pretty much obsolete over the past few years. Big companies like Microsoft have decided to let it die its own slow death. There's also no new libraries becoming available, so its just the matter of time when it will be completely forgotten. Some of the reasons for it could be:

- STM can introduce problems when implementing data structures
- Nesting STM-based data structures within one another can arise conflicts
- It can reduce the cache performance advantages
- Various STM benchmarks have shown that for large datasets with low contention performance can increase when using a good layout, but in single threaded cases with small datasets it has a disadvantage

STM can speed up the performance only under certain conditions, but has a bigger chance to reduce it in general, the same concept goes to using it in games. It can speed up some algorithms, but most of the time its useless and can give a loss of performance.