****

**Institute of Technology, Carlow**

**B.Sc. (Honour) in Software Development**

**CW228**

**Specification**

**Thread Pool Benchmarking**

**Name: Bowen Nong**

**Login ID: C00163585**

**Supervisor: Joseph Kehoe**

**Date: December 14th 2012**

Contents

[1. Vision 2](#_Toc343245933)

[1.1 Introduction 2](#_Toc343245934)

[1.2 Stakeholder Descriptions 2](#_Toc343245935)

[1.3 Summary of project Features 2](#_Toc343245936)

[1.4 Project Overview Diagram 3](#_Toc343245937)

[2. Technology specification 3](#_Toc343245938)

[2.1 OpenMP 3](#_Toc343245939)

[2.1.1 Overview Of How Is Works: 3](#_Toc343245940)

[2.1.2 Is It Easy To Use? 4](#_Toc343245941)

[2.1.3 How Flexible Is It 4](#_Toc343245942)

[2.1.4 How Does It Schedule 4](#_Toc343245943)

[2.1.5 Simple Codes 5](#_Toc343245944)

[2.2 Games Algorithms for benchmark 6](#_Toc343245945)

[2.3 The Benchmark 10](#_Toc343245946)

[ Timing 10](#_Toc343245947)

[3. Conclusion 12](#_Toc343245948)

[4. References 12](#_Toc343245949)

Specification

# Vision

## Introduction

The threads were used to increase the speed of running programs. They allow the computer to work seen like doing job on the same time. Actuary, the CPU just allocate the work time for each thread. When the threads are created or killed by system, the CPU will give the sources and time. If each task needs to create a thread for working, it will waste lots of sources and time. However, on the other way, we created a group of threads for jobs. Then we do not kill the thread. When a thread finishes a task, it will “sleep” in the queue. According to the scheduling algorithms of threads, the thread goes out of queue for the new task. It will re-use the threads and save the time and sources. So the “box” which we use to create, kills and manages the group of threads is thread pool.

Thread pool is managed by the system, so that programmers do not need to spend time taking or cart thread management. And they can concentrate on application tasks. The threads are stored by priority.

According to this project, the thread pool schedule algorithms will be known, and which schedule algorithms are suitable in games. The timing metrics will be used in the project.

## Stakeholder Descriptions

1. Features
   1. Customers and benefits

There are a number of works scheduling algorithms used to improve the performance of multithreaded software. Researching the algorithms of thread pool.

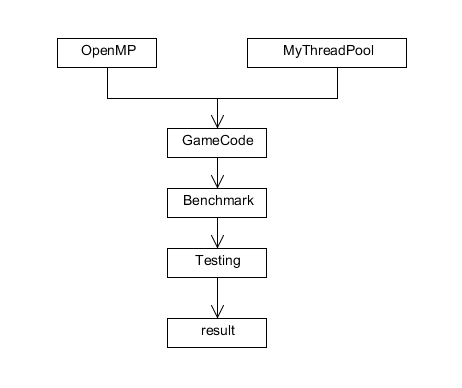
* 1. The type of incident

The project allows thread pool to be implementing the kinds of benchmarks.

## Summary of project Features

* Using OpenMP.
* Difference game algorithms are used to benchmark.
* Testing on benchmark suite.

## Project Overview Diagram

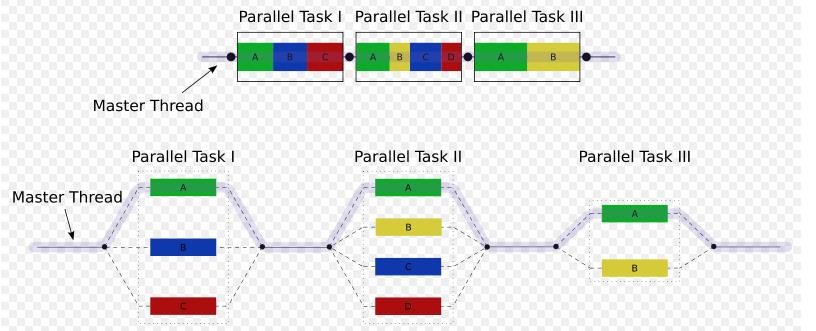


# Technology specification

## OpenMP

### Overview Of How Is Works:

When the OpenMP application begins to work, it should create the main thread. As the program executes, the application can encounter parallel regions in the pool. The thread group will create by the main thread. At the end of a parallel region, the thread teams are parked and the master thread continues execution.

 Example 2: It is the OpenMP model of thread implementation.[OpenMP]

### Is It Easy To Use?

In my point, the answer is yes. The programmers did not need to consider how to run the thread. It takes programmers attention to algorithms. The programmers just use “pragma” apply the mind of which their want system to do.

### How Flexible Is It

The OpenMP give the abstract description of parallel algorithms. The programmers write the standard in source codes, and add the synchronous standard when it necessary. If the compiler did not support the OpenMP, the standards can be ignored.

### How Does It Schedule

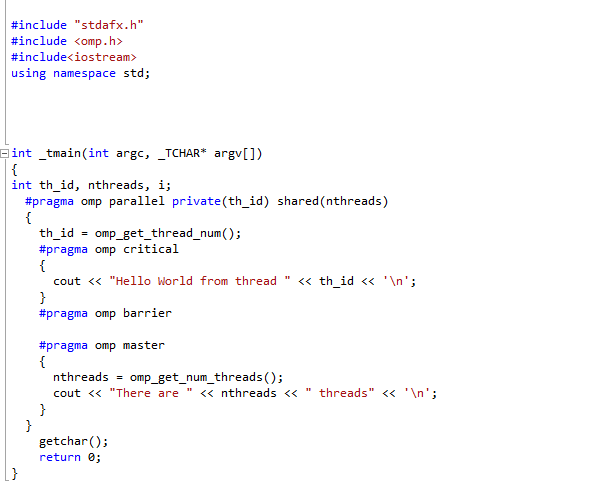
The OpenMP has three kinds of scheduling, which are dynamic, guided and static.

**Static:** all the threads are allocated iterations before they execute the loop iterations. The iterations are divided among threads equally by default. However, specifying an integer for the parameter chunk will allocate chunk number of contiguous iterations to a particular thread.

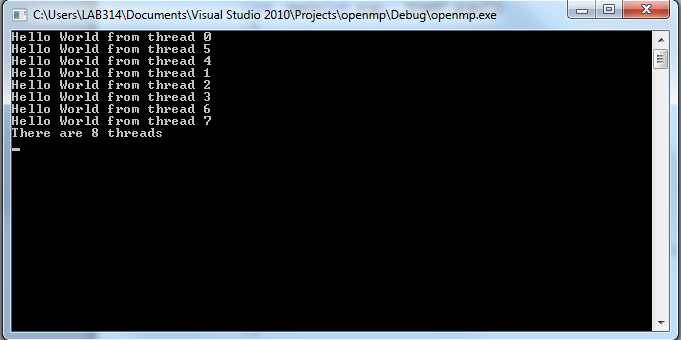
**Dynamic:** some of the iterations are allocated to a smaller number of threads. Once a particular thread finishes its allocated iteration, it returns to get another one from the iterations that are left. The parameter chunk defines the number of contiguous iterations that are allocated to a thread at a time.

**Guided:** A large chunk of contiguous iterations are allocated to each thread dynamically (as above). The chunk size decreases exponentially with each successive allocation to a minimum size specified in the parameter chunk.

### 2.1.5 Simple Codes



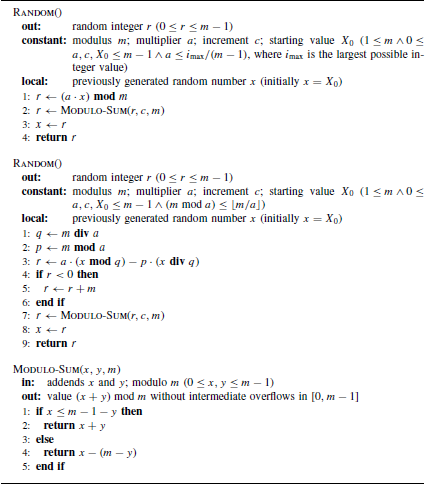
On 8 threads output like this:



Example 3: it shows the simple “Hello World” code of OpenMP. [OpenMP]

## Games Algorithms for benchmark

**Random Numbers:** The random numbers used to generate game outcomes. The outcomes are generated in a highly secure device and cannot be used to determine the correct choice of player selection or influence the game outcome.

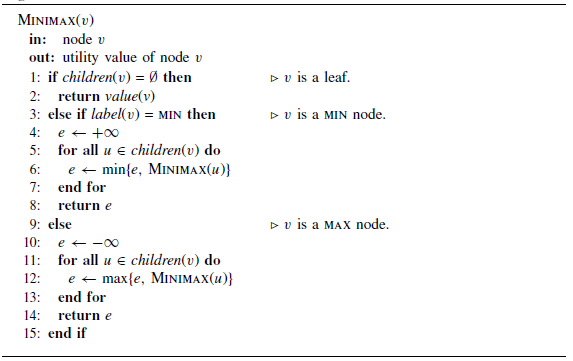


Example 4: They are two algorithm examples of Random Numbers, and describe implementation variants of the linear congenital method defined. [GameAlgorithm]

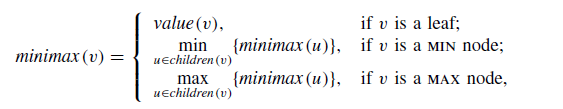
Description:

The first one can be used when a(m − 1) does not exceed the largest integer that can be represented by the machine word. For example, if m is a one-word integer, the product a(m − 1) must be evaluated within a two-word integer. The second variant can be applied when (m mod a) ≤ m/a. The idea is to express the modulus in the form m = aq + p to guarantee that the intermediate evaluations always stay within the interval (−m, m).

**Game Tree:**  A game tree is a directed graph whose nodes are positions in a game and whose edges are moves.



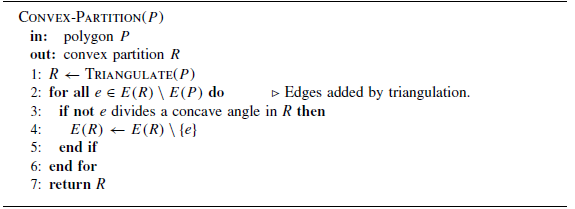
Example 5: It is an Algorithm example of Minimax. [GameAlgorithm]



Example 6.1

The minimax value for a node *v* can be defined with a simple recursion where *children(v)* gives the set of successors of node *v*. Algorithm 4.1 implements this recurrence by determining backed-up values for the internal nodes after the leaves have been evaluated. The Example 16 algorithm runs the equation in Example 16.1.

**Path Finding:** Path Finding in the context of video games concerns the way in which a moving entity finds a path.

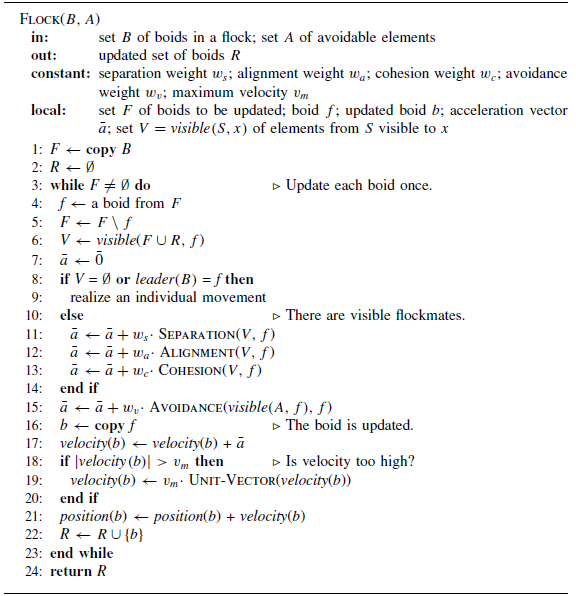


Example 7: It is an Algorithm example of Path Finding. It is Hertel–Mehlhorn method for convex partition. [GameAlgorithm]

Description:

Hertel– Mehlhorn heuristic finds a convex partition in the time *O(n* + *r* log *r)*, and the resulting partition has at most four times the number of polygons of the optimum solution (Hertel and Mehlhorn 1985). The method first triangulates the original polygon.

**Decision-making:** The Decision-making is the set of problems which are strategic, tactical and operational.



Example 8: It is an Algorithm example of decision-making. It is Flocking algorithm. [GameAlgorithm]

Description:

There are four behavior rules in this algorithm. They are following:

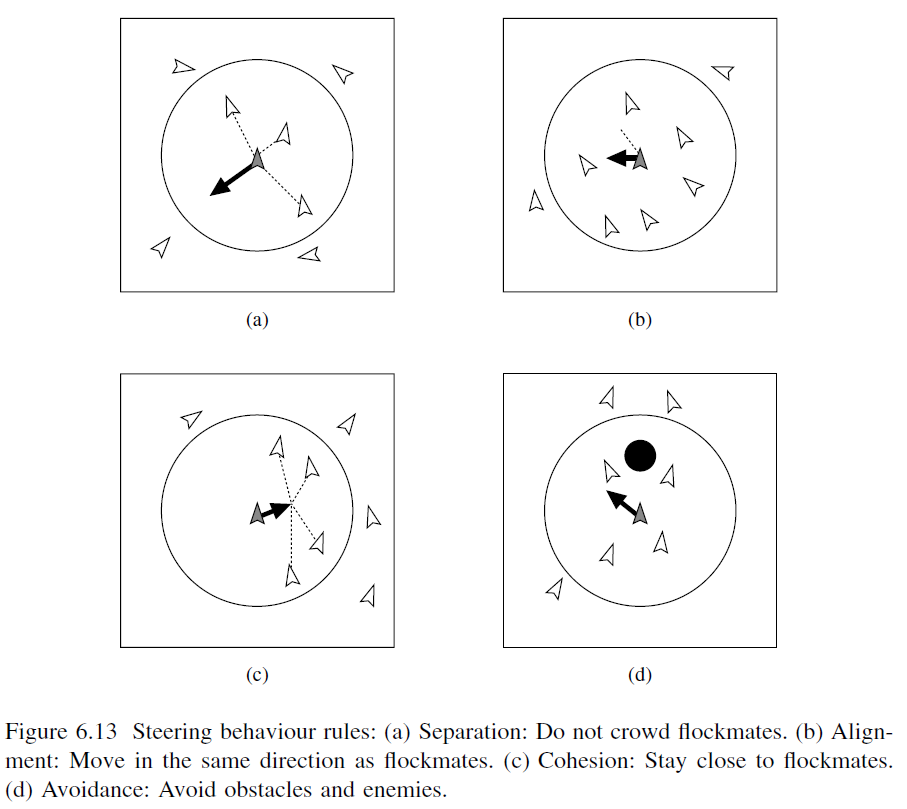
(i) *Separation*: Steer to avoid crowding local flockmates. A boid should maintain a certain

distance from the nearby boids to avoid collisions with them.

(ii) *Alignment*: Steer towards the average heading of local flockmates. A boid should move in the same direction as the nearby boids and match its velocity accordingly.

(iii) *Cohesion*: Steer to move towards the average position of local flockmates. A boid should stay close to the nearby flockmates.

(iv) *Avoidance*: Steer to avoid running into local obstacles or enemies. A boid should escape dangers when they occur.



## The Benchmark

### Timing

timeGetTime()  
returns system time since windows started in milliseconds.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 | //.h  #ifndef \_TIME\_H\_  #define \_TIME\_H\_  #pragma comment (lib,"Winmm.lib")  #define WIN32\_LEAN\_AND\_MEAN  #include <windows.h>  #include <mmsystem.h>  class cTime  {  public:  static cTime\* Instance();  static void Destroy();  void operator()();  UINT64 GetDTime(){return mDTime;}  private:  cTime();  ~cTime();  static cTime \*mInstance;  UINT64 mLastGameTime;  UINT64 mDTime;  };  #endif |

The class is storing the time when then code begin time of running and the end time of running.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 | //.cpp  #include "Time.hpp"  cTime \*cTime::mInstance = 0;  cTime \*cTime::Instance()  {  if(mInstance)  return mInstance;  return mInstance = new cTime;  }  void cTime::Destroy()  {  if(mInstance)  delete mInstance;  }  cTime::cTime():  mDTime(0)  {  mLastGameTime = (UINT64)timeGetTime();  }  cTime::~cTime(){}  void cTime::operator()()  {  UINT64 ticks = (UINT64)timeGetTime();  mDTime = ticks - mLastGameTime;  mLastGameTime = ticks;  } |

Example usage

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 | #include "time.h"  #include <iostream>  using namespace std;  void main()  {  cTime &TimeRef = \*cTime::Instance();  TimeRef();  // ... code to benchmark here ... //  TimeRef();  cout<<"Benchmark: "<<TimeRef.GetDTime();  } |

The codes which are benchmarked will be write in the middle of two TimeRef() functions. At the end of running those codes, it will output the number of time.

# Conclusion

A summary of thread pool benchmark, the project is simple designed. This project will use a thread pool that implements a number of these algorithms. It will need to develop a number of game algorithms benchmark programs that can be used to test these algorithms. The benchmarks will each represent a different type of concurrent problem. Finally, the algorithms will be tested with these benchmarks to compare the different algorithms performance. According to compare each using time, analysis which thread pool schedules are suitable.

# References

[OpenMP] <http://msdn.microsoft.com/en-us/magazine/cc163717.aspx>  
[Assess 24-10-2012]

[GameAlgorithm] Jouni Smed, Harri Hakonen, “Algorithms and Networking for Computer Games”, *University of Turku, Finland, 2006*