

Multithreading in games

Qingli Liu



Multithreading in Games

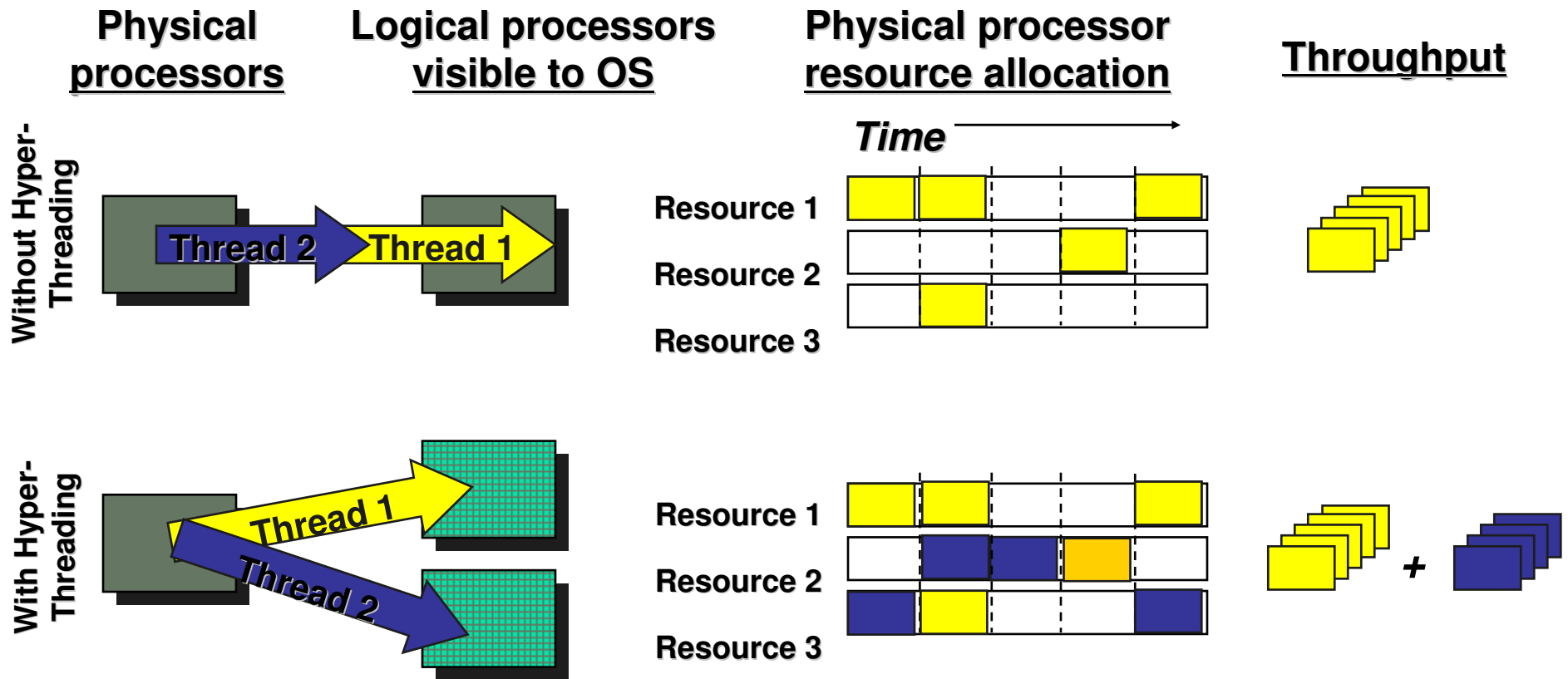
- Motivation
- Introduction
- Multithreading strategies and principles in games
- Implementation of multithreading
- Performance of multithreading
- Summary



Motivation

- Why should you thread your games ?
 - Technical reasons
 - Parallelism is the future of CPU architectures -> easy to scale (HT, multi-core, etc)
 - Do other things while waiting for the graphics card/driver
 - Good MT design scales, and prevents repeated re-writes
 - Business reasons
 - Differentiate yourself in a competitive landscape
 - All PC platforms will support Multi-threading
 - Parallel programming education will pay off with multiple platforms (PC, consoles, server, etc)
 - MT scales more -> extends product lifetime

Introduction - Hyper-Threading Technology



**Higher resource utilization, higher output
with two simultaneous threads**



Introduction - Multiprocessing

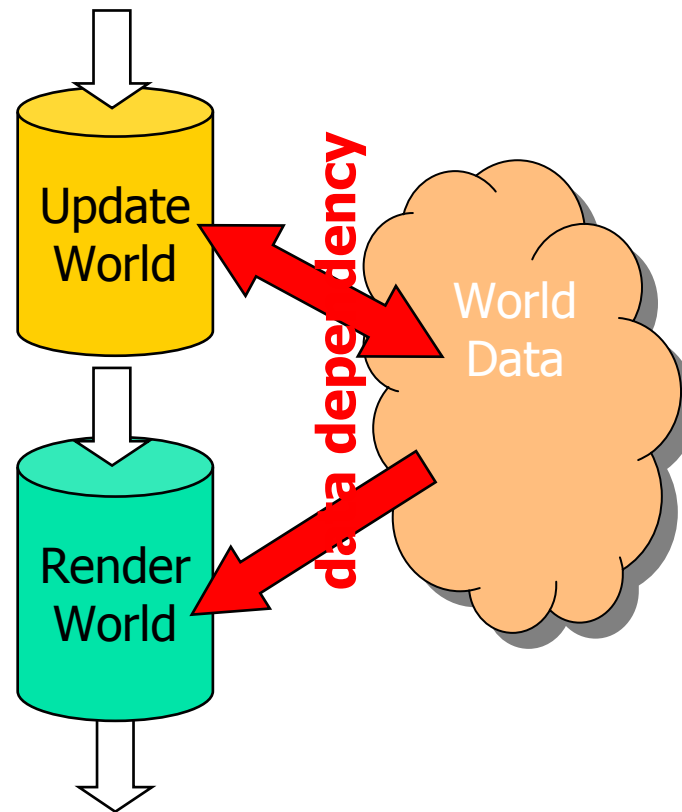
- Multiprocessing systems have multiple processors running at the same time. Traditional multiprocessing systems have anywhere from 2 to about 128 processors. Beyond that number (and this upper limit keeps rising) multiprocessing systems become parallel processors.
- Multiprocessing systems allow different threads to run on different processors. This capability considerably accelerates program performance.
- Now two threads can run more or less independently of each other without requiring thread switches to get at the resources of the processor. Multiprocessor operating systems are themselves multithreaded and they too generate threads that can run on the separate processors to best advantage.



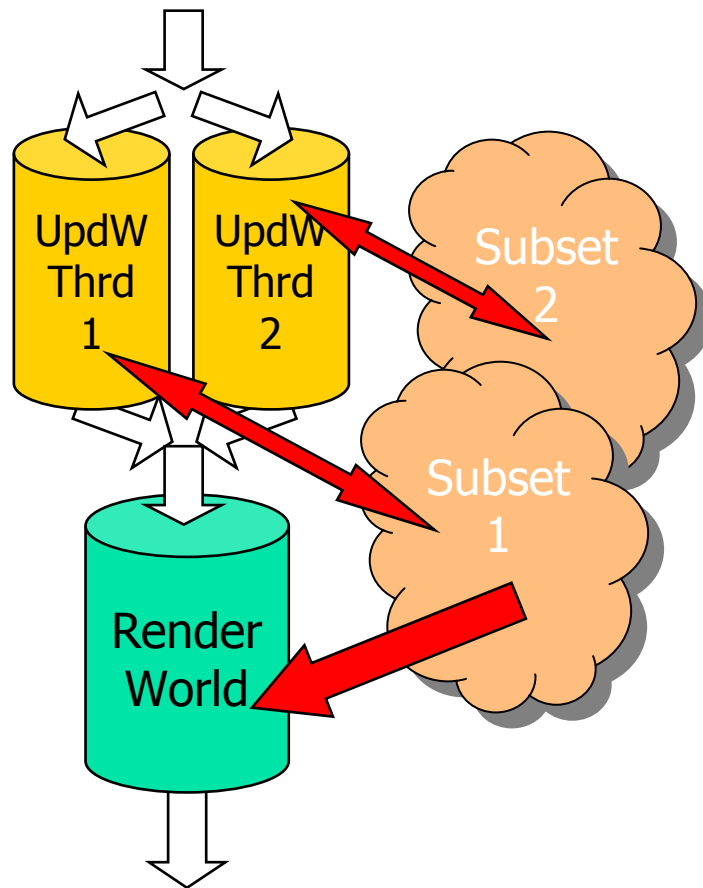
Multithreading strategiey in games

- Utilize Data Parallelism
 - Process disjoint data simultaneously
- Utilize Task Parallelism
 - Process disjoint tasks simultaneously
- Choose task-level or data-parallel threading for various parts of an application

Game's Pipeline Model

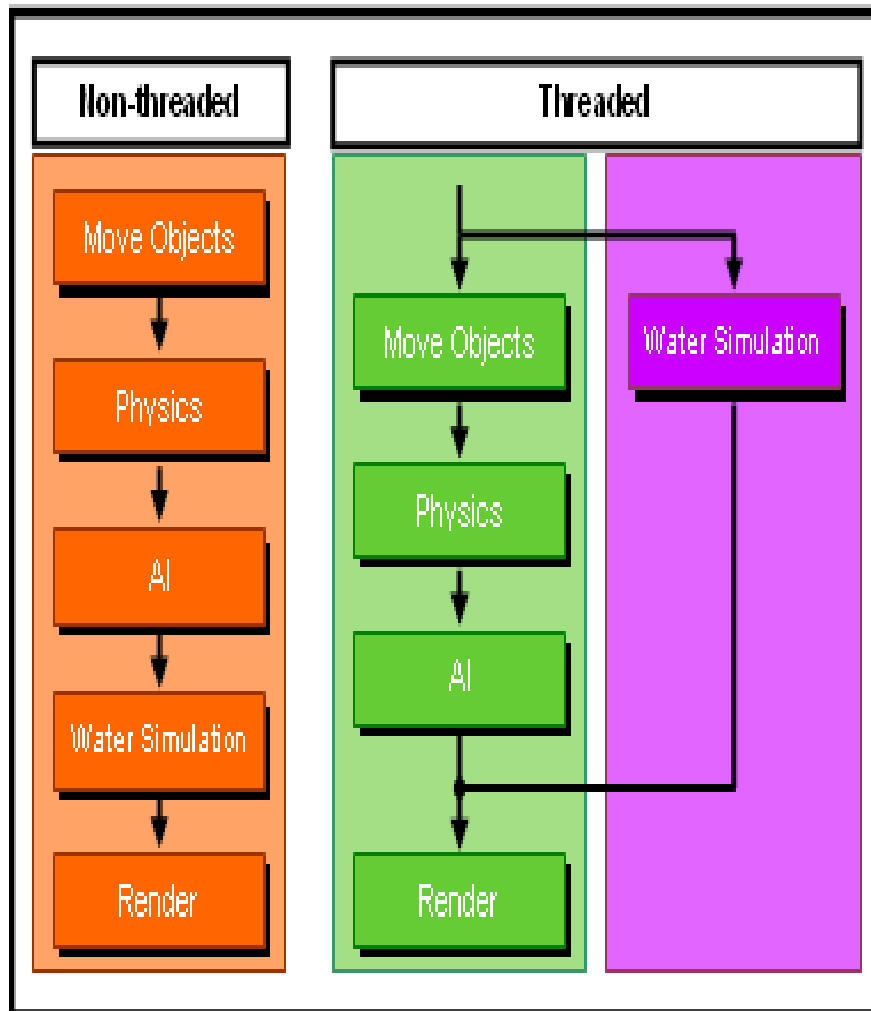


Data Parallelism in Games



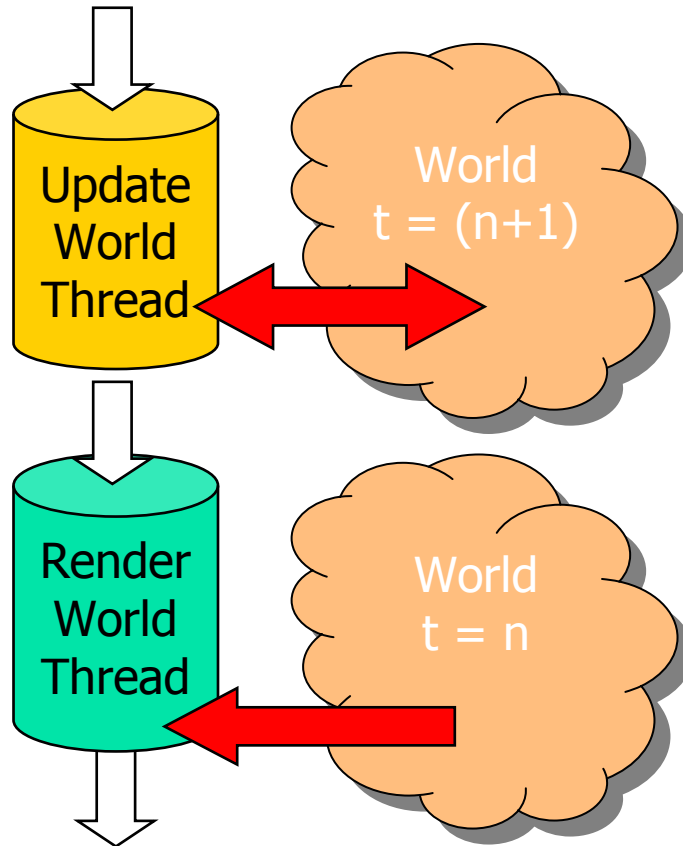
- Threading a single function to operate on two or more blocks of data at the same time.
- Execute tasks on secondary thread
 - Audio processing
 - Networking (including VoIP)
 - Particle Systems and other graphics effects
 - Physics, AI
 - Content (speculative) loading & unpacking
- Multithread Procedural Content creation
 - Geometry, Textures, Environment, etc...
- Threading Potential
 - Good for CPU bound games
 - Easy to implement

Task Parallelism in Games



- Has to do with placing different functional blocks of code on separate threads in order to take advantage of parallelism.
- Thread steps
 - Determine Functional Blocks
 - block flow diagram
 - Determine Dependencies Between Blocks
 - determine whether a grouping of data should be duplicated or whether access to it should be synchronized

Task Parallelism in Games



- Multithread whole 3D Graphics Pipeline
 - Thread 1 = Render Frame (n)
 - Thread 2 = Update Frame (n+1)
- Threading potential
 - Good for GPU bound games
 - Difficult to implement due to dependencies, but not impossible

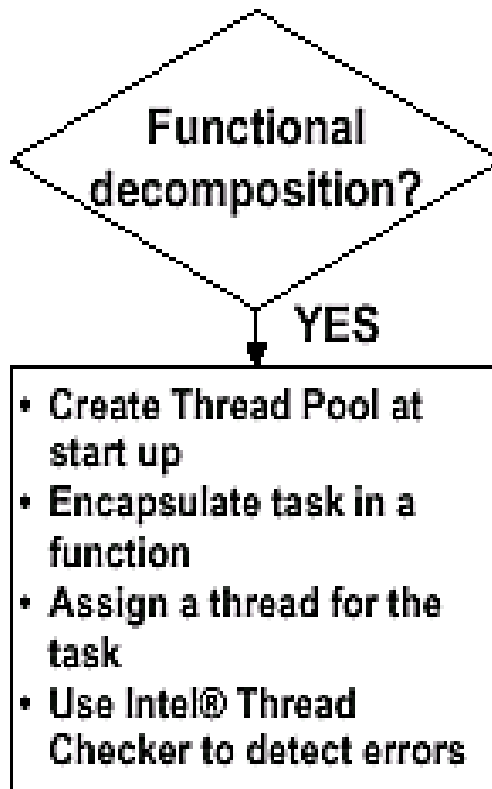


Implementation of Threading for Data Parallelism

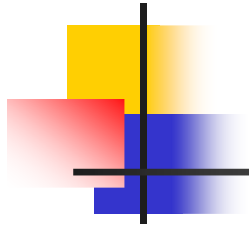
```
#include <stdio.h>
#include <omp.h>
int numIterations = 1000000;
int main()
{
    double x, pi, sum=0.0, step;
    step = 1./((double)num_steps);
    for (int i=1; i < numIterations; i++)
    {
        x = (i - .5)*step;
        sum = sum + 4.0/(1.+ x*x);
    }
    pi = sum*step;
    return 0;
}
```

- Apply threading to Data Parallelism
 - multiple threads need to be assigned to perform the same functionality on different data.
 - OpenMP is recommended

Implementation of Threading for Task Parallelism



- Apply threading to Task Parallelism
 - the roles of different threads are defined by having different functionality (as opposed to identical functionality that is applied to different data).
 - Use thread pools to manage the threading aspect of the problem and distribute the available tasks among the inactive threads.
 - Explicit threading is the recommended solution
 - Thread pool
 - Also as know as a task or work queue
 - Threads are only created once, put to sleep when not used



- API / Library
 - Win32* threading API
 - P-threads

- Programming language
 - Java*

- Programming language extension
 - OpenMP™

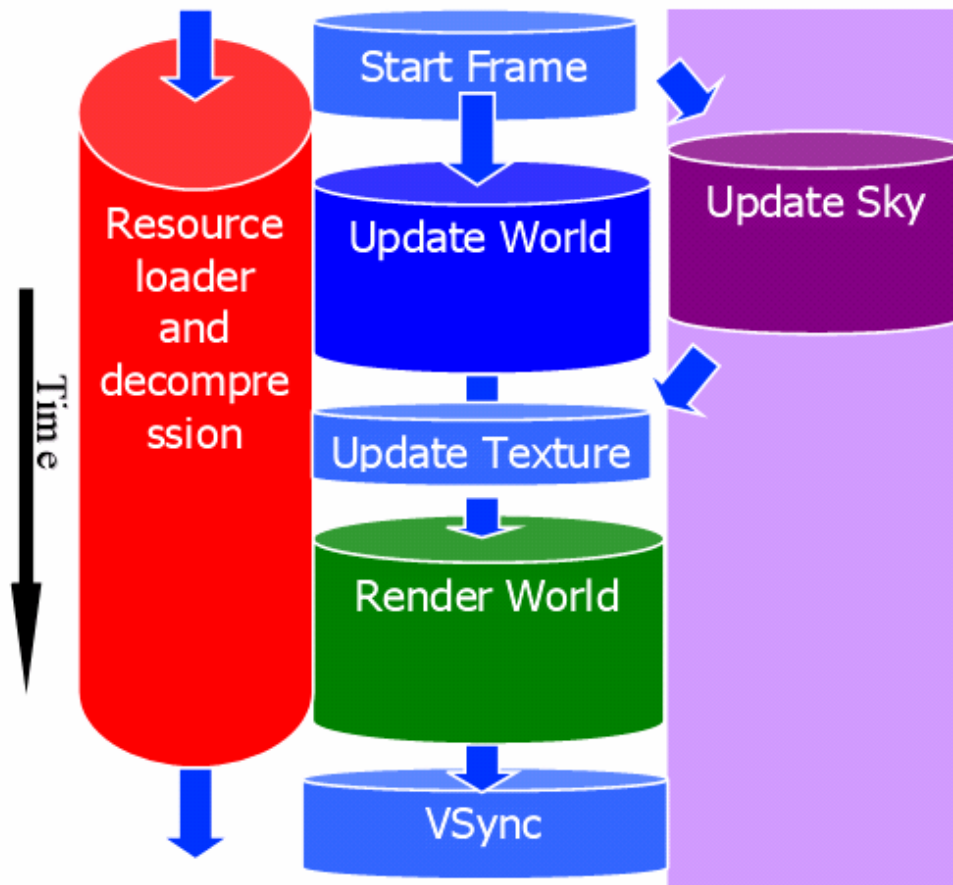
```
My_thrd_func(void* params)
{
    begin, end <- params
    for(i=begin;i<end; i++) {
        a[i] = b[i] * sqrt(c[i]);
    }
}
```

```
// Win32
handle =
    CreateThread(NULL,0,my_thrd_func,
                param,0,NULL);
```

```
// java
Thread myThread = new MyThreadClass( );
myThread.run();
```

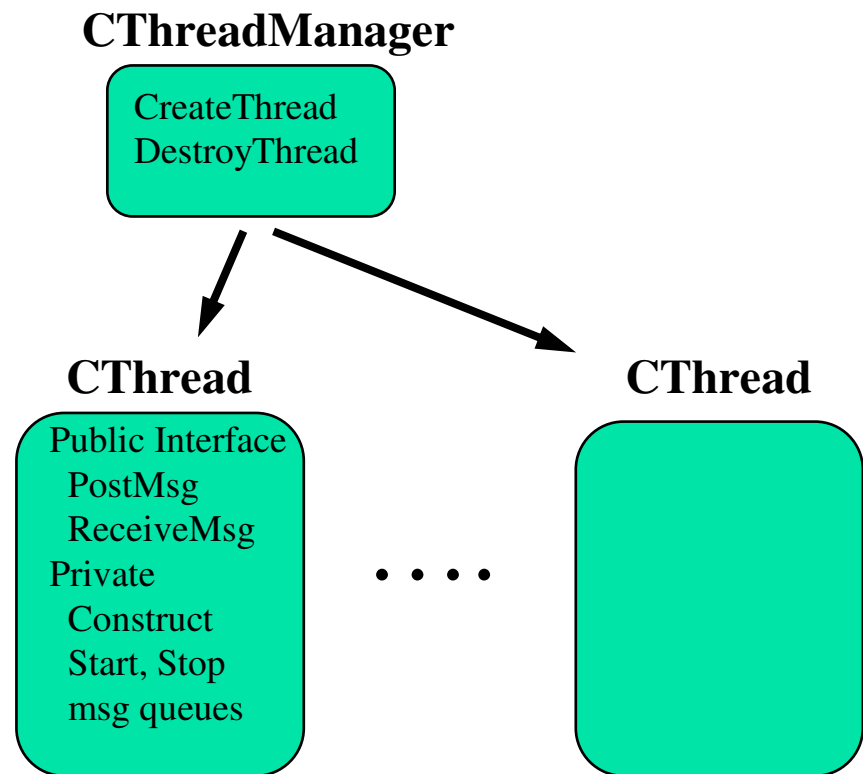
```
// OpenMP
#pragma parallel for
for(i=0; i<max;i++){
    a[i] = b[i] * sqrt(c[i]);
}
```

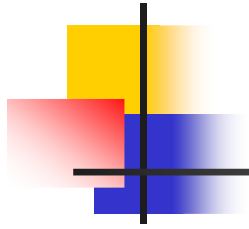
Case Study Lego/Argonaut Bionicle



- Bionicle is a third person action game
- Bionicle tasks were run on different threads.
- The main thread was for Update World and rendering.
- A worker thread was spawned to handle Update Sky.
- File Read and Decompress in Second thread

- Problem: Need a Threading system that is easy to use, object oriented and cross platform
- Solution: Roll our own Thread classes and message passing model.
 - Pros: Max flexibility, full control, and platform indep.
 - Cons: Harder up front, less supporting tools
- CThreadManager
 - Controls lifetime
- CThread
 - Win32





- CThread and CThreadManager encapsulated multi-threading details: synchronization, creation, destruction
- Procedural sky effect easy to add
- Threading streamed IO reduces code complexity.



Performance of Multithreading

- Bad Multithreading is worse than no Multithreading
- Amdahl's Law

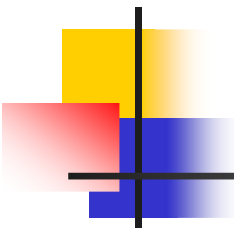
$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

$Speedup_{overall}$ = Overall speedup of application/workload

with enhanced sections of code.

$Fraction_{enhanced}$ = Portion of code that has been enhanced or

made parallel. ($0.0 \leq Fraction_{enhanced} \leq 1.0$)

- 
-
- Apply Amdahl's Law to Single-processor (UP) system, Hyper-Threading Technology system (e.g. Speedupenhanced=1.3) , Dual-processor (DP) system (e.g. Speedupenhanced=2)
 - Use performance anylyser tool e.g. Intel® VTune™ Performance Analyzer
 - Profile the game early and often
 - Helps build an incremental view of performance



Summary

- Thread your Game – it can be done!
- As the number of processors in desktop systems increases, threading will grow more and more important
- Start multithreading as early as possible, ideally in code design stage
- Choosing the right threading method minimizes the amount of time spent modifying, debugging, and tuning threaded code.
- Save time – Use Threading Tools to maximize threaded performance