Volume 2, No. 3, May-June 2011
ISSN No. 0976-5697
International Journal of Advanced Research in Computer Science
RESEARCH PAPER
Available Online at www.ijarcs.info

Performance Analysis of Gaming Application on Multicore Architecture

N. Muhammed Talha*
School of Computing Science and Engineering
VIT University
Vellore, India
thal_n@yahoo.com

M. Rajasekhara Babu
School of Computing Science and Engineering
VIT University
Vellore, India
mrajasekharababu@vit.ac.in

M. Khalid
School of Computing Science and Engineering
VIT University
Vellore, India
mkhalid@vit.ac.in

Abstract*:* In this paper we are describing the performance of gaming application using a multi-threaded gaming engine (smoke). That is designed to scale to as many processor cores are available within a system. It does this by executing different functional blocks in parallel so that it can utilize all available cores/processor. Here we changing configuration files with (no recompilation) to modify the way the existing technology. The framework is designed to allow the system(AI, Physics) to talk each other in a efficient threaded manner, without writhing custom code. This project aims to perform parellelizing technique over the gaming code to achieve a higher level of performance.

Keywords: Multithread, Smoke engine, Artificial Intelligence, Physics, and configuration files.

I. INTRODUCTION

Multi-core processors are widely used across many application including general-purpose, embedded, network, digital signal processing (DSP), and graphics. Several business strategies drive the development of dual-core architectures. For decades, it was possible to improve performance of a CPU by decreasing the area of the integrated circuit, which was drive down the cost of the IC [1]. Alternatively, for the same circuit area, more transistors could be utilized in the design, which increased functionality, especially for CISC architectures. Eventually these techniques reached their limit and were unable to improve CPU performance. Multiple processors had to be used to gain speed in computation. Multiple cores were used on the same chip to improve performance, which could then lead to better sales of CPU chips which had two or more cores. Intel has invented a 48-core processor for research in cloud computing [1]. The largest boost in performance will likely be noticed in improved response-time while running CPU-intensive processes, like antivirus scans and more. For example, if the automatic virus-scan runs while a movie are being watched, the application running the movie is far less likely to be starved of processor power, as the antivirus program will be assigned to a different processor core than the one running the movie playback [2]. Now we are trying to implement the gaming application in multicore architecture. We used a game engine named as smoke on multi core architecture. Smoke is a tech demo using the parallel architecture framework. As can be seen this is not a simple demo, the developers wanted to produce a framework that can be used for game technologies and can fully incorporate middleware such as Havoc, FMOD, etc. Another goal was the need to be partitioned and configurable, allowing game developers to tweak and adjust the workload. The primary goals for developing this framework were:

1. Performance; this example shows an 8-core system, but the framework was designed from the ground up to expand beyond 8-threads, to ensure complete loading of all the CPUs.

2. The framework was developed to allow the exploration of other game technologies by getting up the data in such as way to support ease of prototyping and to be able to incorporate new technologies [1]. Many things can be threaded, but to be able to plug-in new functionality, the game framework must be structured in such a way to allow efficient communications in a threaded environment, without undue performance bottlenecks.

The framework is setup to be totally configurable and partitioned. The framework was designed to allow the Systems (AI, Physics) to talk each other in a efficient threaded manner, without writing custom code for communications between each of the Systems. The primarily reason the framework was developed was to teach. The smoke engine are also used multi-threading concepts to achieve performance over game The developers wanted to develop a parallel architecture framework that shows efficient threading and make it easy to prototype and to add new technologies. From these we understand how change synchronization is handled in the parallel architecture framework and how it contributes to the scalable game framework and Recognize how common interfaces are used to interact between Systems (AI, Physics, Graphics, etc) [3].

II. ELEMENTS OF GAME PERFORMANCE

A game is an interactive real time application which takes input from the user, performs some computations and

displays the game in a fixed time, usually aiming at rendering at a minimum illusion of a continuous motion. It used to reduce in order to avoid the frame rate from dropping below this frequency [5]. Measures of game performance include i. the fill limit, which indicates how fast the application can fill polygons to form solid surfaces; ii. the texel limit, which indicates how fast texel, which are texture elements, can be mapped onto polygons; iii. the polygon limit, which indicates how fast the game's primitives can be processed, including geometry operations such as scaling, translation, and rotation; iv. depth complexity, which reflects how many polygons are positioned one behind the other, causing the Z buffer to suffer when there are too many polygons behind each other; v. CPU load composed of scene management, score accounting, as well as artificial intelligence and physics. When one or more of the measures exceed a certain limit, the scene cannot be rendered and transferred to the frame buffer on time, so a game update is missed and a frame isdropped. The game engine contains the following loop

```
While (game has not ended)
{ UpdateGame();
RenderGame(;
}
```

where UpdateGame gets the user inputs, performs the computations necessary for updating the game's state, artificial intelligence, physics, and audio, while render Game focuses on displaying the game, including geometry/vertex and pixel operations . Typically, to let the game run adequately on both fast and slower hardware, the UpdateGame is placed under an inner loop that controls the game update at a constant rate about 25 times per second, and RenderGame is called with an interpolation argument when called between game updates to give the illusion that the game is running at a high frame rate . This will work adequately on fast and slower hardware we are trying to consider measures of game performance FPS and CPU usage [5].

III. RELATED WORK

In 1980s computers were used to do one thing at a time, and that was word processing, creating spreadsheets or more. These days, users expect to run several applications at a single time. Despite these growing demands, the programmer's role has retained the same – to create a single series of instructions for a system's processor to execute in sequential [2]. CPU vendors have tried to meet the challenges posed by multi-tasking users by producing faster processors, which have the capability to give the user the impression that multiple applications are running at once. What drove the industry to multicore technology wasn't just the need for more processing speed, less heat because the fastest chips were heating up faster than the average fan could cool them down, and more energy efficiency because single-core chips rely on tightly packed transistors to get the job done [3].

Compared to several single-core chips, a multi-cor e chip is easier to cool because the CPUs are simpler and use fewer transistors. It means they use less power and dissipate less heat overall. As for performance, each multi-core processor can work on a different task at the same time. Parallel processing used to require more than one chip or clever algorithms to simulate parallel processing from the software side. In a multi-core processor, parallelism is already built in [3]. The problem is that most programmers—truly, most humans— think sequentially, so most codes are written to run sequentially. Parallelizing them can require heroic effort.

As a reader of Computer, you probably don't need the detailed explanation of why most of the industry has shifted to multi-core technology. But in all honesty, many embedded system designers are still struggling to determine whether multi-core really buys them anything in terms of performance. Resolving this type, requires a thorough understanding of the target application problem, the characteristics of multi-core processors that could be used, and the amount of time that must be invested to make the transition [3]. Having reliable performance information provides a good starting point for analyzing these type of factors, but. In other words, it's imperative to take the right types of benchmarks to accurately predict the performance of the multi-core processor [2].

Many performance evaluation techniques and tools have been developed in the past but most performance tools support single-threaded applications. Recently, Intel started to offer performance tools that support multi-core and multithreading, but they require special hardware performance monitoring support. The complexity of the systems increased and created significant challenges to the programmers. To become a good application developer for Today's *different* multi-core systems, one will have to be familiar with multiple Instruction Set Architectures (ISA's) on the processor in the platforms in order to optimize for application performance [3].

IV. PROPOSED WORK

The steady of symmetric multiprocessing to putting many functional units on a chip to multi-core has been a long time approach. Software ran faster year after year, not because of software innovations, but because chip makers kept adding transistors to the standard single-processor architecture. If we want faster speeds, we have to embrace concurrency and make use of multiple processors on the chip at once. To ease the complexity on the developer, performance analysis tools are essential, and they provide significant help to the developers on exploring and tuning the application performance [3].

Faster processors won't give faster games unless complex code can be broken down into smaller blocks. Having reliable performance information provides a good starting point for analyzing these factors, but "reliable" is the operative word. In other words, it's imperative to select the right types of benchmarks to accurately predict the performance of the multi-core processor.

Performance evaluation is key to many computer applications. Many techniques and profiling tools are available for measuring performance, but most of them depend on the hardware and the software on which they run. For a new platform, or a platform which is not popular, programmers usually suffer from few analyses tools, which has been a critical problem for application development on many systems. Thus, a performance analysis tool with the

software mechanism is quite important for developing applications [5].

During the past two decades, benchmarks that only exercised a processor core's internal workings were sufficient. In fact, these benchmarks are still valid depending on the processor characteristics. Although these benchmarks can run on top of the operating systems. Performance is measured in iterations per second, where iteration is the sequential execution of the benchmark. The compiler also plays a big role in this type of benchmarking. In fact, we've seen as much as 70 percent performance difference depending on the compiler used to generate the benchmark results [4].

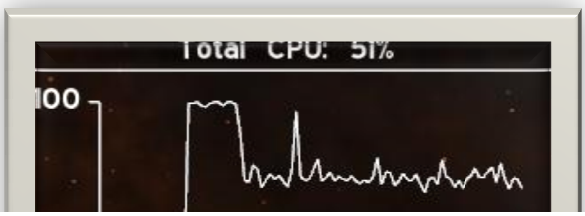
V. IMPLEMENTATION

OpenMP applications programming interface provides various compiler directives, runtime routines and environment variable which provide capability to parallelize a serial program. So by using OpenMP API we can parallelize execution of program between two or more cores in dual or multi-core machines respectively and improve the performance in games [2]. This project was executed on a much accelerated schedule and was intended to assess the programmability of the multi-core architecture. To that end, the team started with an established code base that included a general game database, game engine. These challenges resulted in a quick difference between the single and multi-core code base. In order to create a stable, robust reference system, the team decided to develop the game on a single-core first. Implementing an application for the multi-core architecture requires that programmers design for one of its key architectural features [4]. Practically, this was not possible, given the time constraints of the project. We focused our efforts on porting the code that represented the bulk of the workload. Because the application relied heavily on rigid body dynamics to provide interesting game play. Either task-level or data-level parallelism could have been employed across the architecture.

VI. PERFORMANCE

Performance
6 FPS
1 thread
CPU 1: 12 %
CPU 2:86%

Table : Performance using single thread



Graph: Using Single Thread

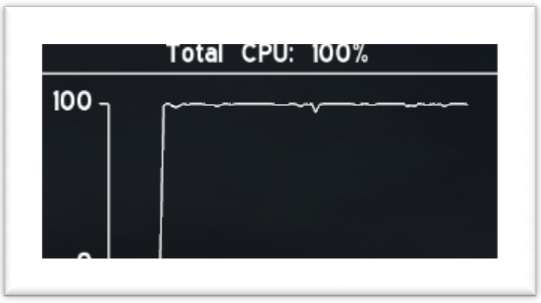
Performance rely on FPS and CPU usage, Here we are mentioning the performance attained by single-thread processor. These Table 5.1 obtained by using single thread processor by running the game. Using single core processor you cannot attain more performance. See the graph 5.1 below, these are the performance attained by single thread.

Performance
10 FPS
2 thread
CPU 1:94%
CPU 2:98%

Table : Using multithreading

These graph 5.2 obtained by using multi thread processor to run the game. Here we used only two threads. Using two threads you can achieve better performances compare to the single thread. See the graph for the performances It showing the CPU usage as 96% The more core you use, better performance you will get.

Graph: Using Multi-thread



VII. CONCLUSION

This project has described the concept of multi threading by using multi core processor. Here it is shown that how to effectively and efficiently implement the gaming engine using multi-core systems and OpenMP API, extracting as much as parallelism as possible from the algorithm in parallel implementation approach. It also includes the extensive quantitative evaluation of performances both in sequential and parallel implementation. Implementation result shows that, multi-threading using these gaming engine using dual core or multi-core machines provides more performances.

In future this project can be extended for quad-core, 8-core and multi-core processor machines. This project can also be extended to attain more CPU usage.

VIII. ACKNOWLEDGEMENT

I owe a great many thanks to a great many people who helped and supported me during the project. My deepest thanks to professor, [Prof.M.Rajasekhara Babu] the Guide of the project for guiding and correcting various documents of mine with attention and care. He has taken pain to go through the project and make necessary correction as and when needed. I express my thanks to the Director of SCSE, VIT UNIVERSITY & VELLORE, for extending his support. I would also thank my Institution and my faculty members without whom this project would have been a distant reality.

IX. REFERENCES

[1] Pam Frost Gorder, "Multicore Processor for science and engineering", *Computing in science and engineering,* vol:9, Issue:2, 2007, pp: 3-7.

[2] Sangani.K, "computing-Two good to be true- Multicore microprocessor may be great for doing lots of things at once,but what about doing one thing more quickly ?", *Engineering &Technology,* Vol:2, Issue-1, 2007, pp:40-43

[3] Gal-On, S.; Levy, "Measuring Multi-core performance", *Computing in science and engineering,* vol:11, Issue:2, 2007, pp: 2-7.

[4] D.Amora,B; Nanda,A.; Magerlein,K.; Binstock, A.;Yee,B "High Performs Server system and the next generation of online games ", *IBM System Journal* , Vol;45, Issue :1, 2006, pp:103-118.

[5] Sibai F.N, "On making Intelligence Performance Inconspicous in 3D games ", *Innovation in information technology* **,** 20 Nov.2007, pp : 297.

