Research Paper

Journal of Aerospace System Engineering
Vol.13, No.5, pp.1-8 (2019)

ISSN 1976-6300
EISSN 2508-7150
http://dx.doi.org/10.20910/JASE.2019.13.5.1

ARINC 653 멀티코어 기반 그래픽스 렌더링 엔진 분산처리방안 연구

정무경[1,†]

[1]한화시스템

The Study of Distributed Processing for Graphics Rendering Engine Based on ARINC 653 Multi-Core System

Mukyoung, Jung[1,]

[1]Hanwha Systems

Abstract

Recently, avionics has been migrating from a federated architecture to an integrated modular architecture based on a multi-core to reduce the number of systems, weight, power consumption, and platform redundancy. The volume of data which must bo provided to the pilot through the display device has increased, because an integrated single device performs multiple functions. For this reason, the volume of data processed by the graphic processor within a fixed operation period has increased. In this paper, we provide a multi-core-based rendering engine in to perform more graphics processing within a fixed operation period. We assume the proposed method uses a multi-core- based partitioning operating system using the AMP (Asymmetric Multi-Processing) architecture.

초 록

최근 항공기 운용 장비는 시스템 수 및 무게, 전력 소비량 플랫폼 중복성을 줄이기 위해 기존 연합형(Federated) 구조에서 멀티코어를 이용한 모듈형(Integrated Modular) 구조로 변화하고 있다. 이러한 변화로 다수의 기능을 하나의 모듈에 통합함에 따라 디스플레이 장치를 통해 조종사에게 제공해야할 정보량이 증가하게 되었으며, 이로 인해 시스템 운용주기 내에 처리해야할 그래픽양이 증가하게 되었다. 본 논문에서는 멀티코어 시스템을 운용하기 위한 AMP(Asymmetric Multi-Processing) 방식의 파티셔닝 운용 시스템(Partition Operating System)에서 항공기 시스템의 운용주기 내에 보다 많은 그래픽을 처리하기 위해 단일코어로 처리되는 기존의 그래픽스 렌더링 엔진 구조를 멀티코어로 분산하여 처리할 수 있는 그래픽스 렌더링 엔진 분산처리방안을 제시한다.

Key Words : Partition Operating System(파티셔닝 운영체제), Graphics Rendering Engine(그래픽스 렌더링 엔진), Distributed Processing(분산처리), OpenGL SC(오픈 그래픽스 라이브러리), Queuing Port(큐잉 포트), Shared Memory(공유 메모리), Operation Period(운용주기)

1. 서 론

최근 항공기, 자동차 등을 포함한 내장형 탑재 시스템들은 탑재 장비의 크기, 무게, 전력 등의 문제로 기존 다수의 장비가 탑재되어 연동되는 방식인 연합형(Federated) 구조에서 단일 하드웨어 내부에서 모듈단위로 구분되는 모듈형(Modular) 구조로 개발되고 있다[1]. 항공전자분야에서는 모듈형 구조를 지원하고 소프트웨어의 실시간 요구사항 만족 및 안정성을 보장하기 위한 표준규격으로 ARINC 653과 같은 파티션 개념이 개발되었으며[2], 국내에서는 ARINC 653 표준을 따르는 파티셔닝 운영체제를 사용한 개발 및 연구

Received: Sep. 14, 2018 Revised: Aug. 05, 2019 Accepted: Sep. 26, 2019
† Corresponding Author
Tel: +82-031-8020-7786, E-mail: mk.jung.@hanwha.com

되고 있다. 파티셔닝 운영체제는 유닛(Processor/Core)상에 업무에 따라 여러 개의 파티션으로 운용될 수 있으며, 파티션 간의 실시간 동기화를 기반으로 동작하는 구조이며, 개별 파티션은 자체 업무수행과 상대 파티션의 업무 수행결과를 제한 시간 내에 검토할 수 있다. 단일 처리 유닛(Single Processor/Core)상에 하드웨어 추가 없이 업무 수행을 위해 다수의 파티션을 운용할 경우 유닛의 실행 시간을 나누어 순차적으로 실행되기 때문에 파티션간 실시간 동기화의 불가능에 따르는 제약이 발생하게 된다. 이와 같은 제약을 극복하기 위해서는 단일 처리 유닛을 넘어선 멀티코어 또는 멀티프로세서를 이용한 파티션 개념이 적용되었으며, 최근 경향을 보면, 단일 프로세서에 다수의 코어를 탑재하여 저전력 성능을 향상시키는 방식인 멀티코어 기반 AMP 방식의 파티션을 적용하고 있다. 멀티코어 기반 AMP 방식에서는 코어별 별도의 운영체제가 마치 독립된 시스템처럼 동작하게 되며, 공유 자원을 제외한 일반 자원들은 특정 코어에 귀속되게 된다. 이와 같은 기능을 제공하는 상용 운영체제로는 Green Hills 사의 Integrity-178 TuMP와 Wind River 사의 VxWorks 653 3.x가 있다. 항공전자분야에서는 멀티코어 환경에서 안정성을 보장하기 위해 멀티코어 기반 AMP 방식의 파티셔닝 운용시스템을 적용하여 업무에 필요한 다양한 정보를 그래픽을 통해 조종사에게 제공하고 있으며, 조종사에게 제공하기 위한 그래픽 생성은 그래픽스 국제 표준 제정 크로노스 그룹(Khronos Group)에서 안전이 필수로 요구되는 시스템을 위해 제정한 OpenGL SC(Safety Critical)[3] API를 사용하고 있다.

현재 그래픽을 통한 정보 제공은 시스템의 모듈형 구조로의 변화에 따라 Figure 1에서 보는 봐와 같이 여러 개의 그래픽스 장비를 하나의 대화면으로 통합하는 추세이다.

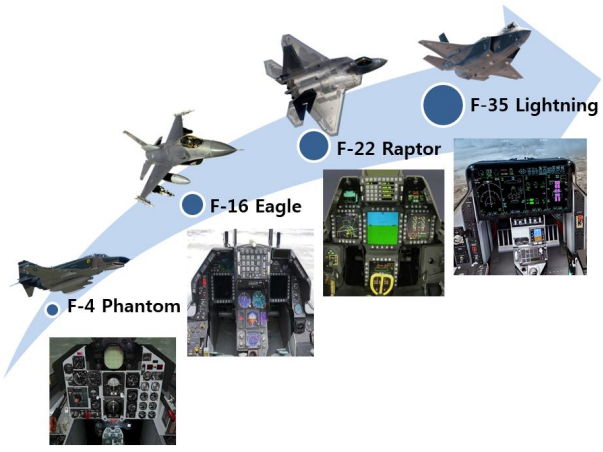

Fig. 1 Avionics Display Integration

이와 같은 대화면화로 항공기의 시스템 운용주기 내에 처리할 그래픽양이 증가하게 되었으며, 멀티코어 기반 AMP 방식에서 특정 코어의 사용량이 높아지게 된다. 이는 운용주기를 통한 실시간 동기화가 불가능해 질수도 있다.

본 논문에서는 멀티코어 상에 파티션 개념을 적용하여 운용할 수 있는 운영체제(ARINC 653 표준)인 VxWorks 653 3.x의 AMP 운용환경에서 그래픽 처리에 따른 특정 코어의 사용량을 분산하여 항공기 시스템 운용주기 내에 보다 많은 그래픽을 처리할 수 있도록 기존 단일 파티션에 운용되는 그래픽 처리 엔진을 두 개의 파티션으로 분산처리하기 위한 기법을 제안하고자 한다.

2. 본론

2.1 배경지식

2.1.1 파티셔닝 및 지원 운영체제

파티셔닝이란 여러 어플리케이션들을 각각 개별 파티션 개념으로 분리하여 각 파티션에 대하여 시간, 공간적 분할을 제공하는 개념을 의미한다. 공간분할은 다른 파티션들이 서로의 물리적 메모리 자원에 영향을 끼치지 못하도록 고립시키는 것을 말하며, 시간분할은 파티션에 할당된 시간을 다른 파티션에서 간섭할 수 없도록 구분하는 것을 의미한다. 항공전자 분야에서는 파티셔닝을 위한 표준을 제정하였으며, ARINC 653이

대표적인 표준이다. ARINC 653 표준을 따라 개발된 운영체제는 파티셔닝의 신뢰성을 보장 받을 수 있다. 다음 Figure 2는 ARINC 653을 지원하는 Wind River 사의 VxWorks 653 구조도를 나타낸다.

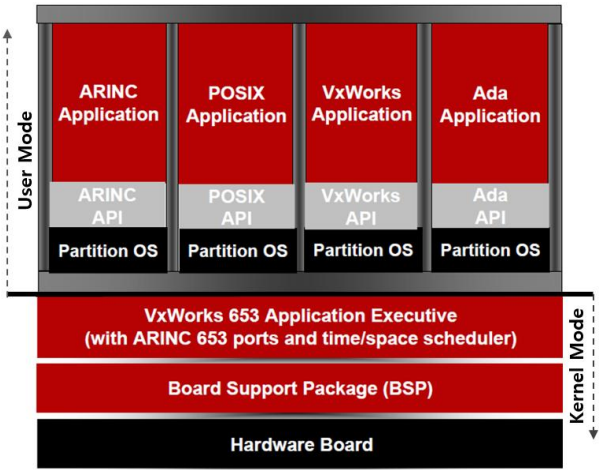

Fig. 2 VxWorks 653 Architecture

위 Fig. 2에서 볼 수 있듯이, VxWorks 653은 하드웨어 상에 ARINC 653 서비스를 제공하며, Partition OS를 이용하여, 시공간적으로 분리된 응용 어플리케이션 운용을 가능하게 하는 운영체제이다. 최근에는 멀티코어 기술이 발전함에 따라 멀티코어 플랫폼 상에서 코어별 독립 운영체제 및 어플리케이션을 동작시킬 수 있는 파티션 운영체제들이 등장하였으며 가상화 기술인 하이퍼바이저(Hypervisor)가 적용되어 파티션별로 다양한 게스트 운영체제의 사용이 가능하다.
다음 Figure 3은 멀티코어 하드웨어 기반에서 코어별 다중 파티션 운용이 가능한 VxWorks 653 3.x의 구조도를 나타낸다.

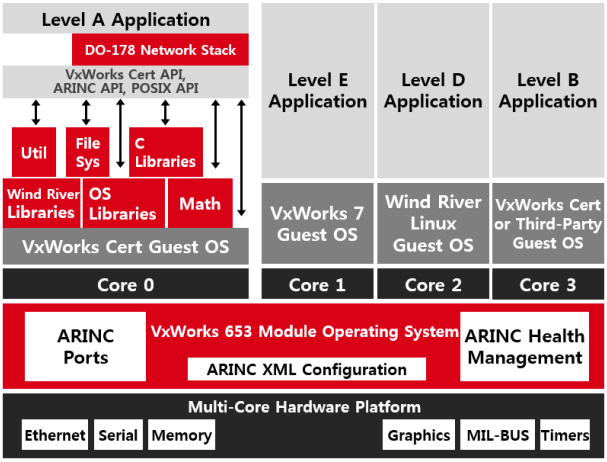

Fig. 3 Integrated Modular Avionics(IMA) with VxWorks 653 Multi-core Edition

위의 Fig. 3와 같이 VxWorks 653 3.x는 멀티코어 하드웨어 상에서 코어별 파티션 환경을 구성할 수 있으며, 단일 코어상에 다중 파티션 환경을 구성하는 것 또한 가능하다. VxWorks 653 Module Operating System으로 표기된 레이어가 하이퍼바이저 역할을 수행하며, ARINC 653 표준이 제공하는 서비스를 제공한다.

2.1.3 그래픽스 렌더링 엔진

그래픽스 렌더링 엔진은 점, 선, 도형을 사용하여 컴퓨터에서 도출되는 다양한 디지털 정보를 시각화할 수 있다. 이러한 그래픽스 처리를 위한 국제표준으로는 Khronos Group에서 제정한 OpenGL[4], OpenGL ES (for embedded systems)[5], OpenGL SC (safety critical)가 있으며, 이 중 OpenGL SC는 항공, 국방, 의료, 자동차 등의 안전이 필수로 요구되는 분야의 2차원 및 3차원 그래픽 출력 및 제어를 위해 제정된 국제 표준이다. OpenGL SC는 지금까지 버전 1.0, 1.0.1, 2.0이 배포되었으며, 국내 항공전자 분야에서는 대부분 그래픽 처리를 위해 OpenGL SC 버전 1.0.1을 사용하고 있다. 아래 Figure 4는 Khronos Group의 OpenGL SC 로드맵과 사용 분야[3]를 보여준다.

Fig. 4 Khronos Group OpenGL SC Roadmap

OpenGL SC 버전 1.0.1(이하 OepnGL SC)[6]은 구현 및 안전 인증 비용을 최소화 하였으며, 항공전자 및 자동차 계기판 디스플레이와 같은 안전에 중요한 응용 분야의 요구 사항을 충족하도록 설계된 101개의 API를 제공한다.

OpenGL SC 렌더링 엔진은 GPU를 사용하여 구현되며 GPU는 Figure 5와 같은 파이프라인[3]으로 동작한다.

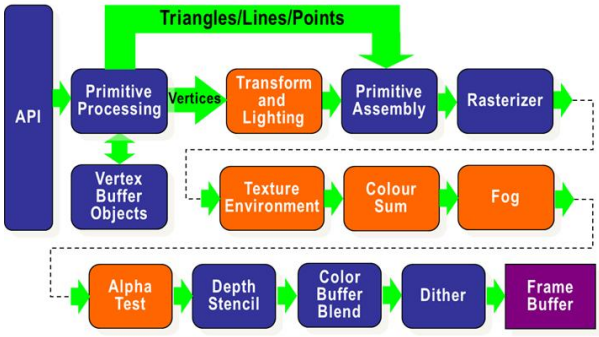

Fig. 5 Graphics Fixed Pipeline

렌더링 엔진은 원하는 기능을 수행하기 위해 각 파이프라인의 기능을 활성화 또는 비활성화 하며, 파이프라인 수행에 필요한 리소스를 관리하고 GPU로 전달한다. GPU로 전달된 리소스는 각 파이프라인 단계를 거쳐 최종 이미지 데이터로 변환되어 프레임버퍼에 기록된다. 프레임버퍼의 데이터는 사용자의 요청에 의해 디스플레이 장치로 전달된다.

2.2 ARINC653 기반 OpenGL SC 렌더링 엔진

2.2.1 기존 단일코어 기반 렌더링 엔진 분석

멀티코어 기반 AMP 방식에서는 코어별 별도의 운영체제가 마치 독립된 시스템처럼 동작하게 되며, 공유 자원을 제외한 일반 자원들은 특정 코어에 귀속되게 된다. 따라서 그래픽영상을 시현할 수 있는 GPU는 특정 코어에 할당되어 해당 운영체제에서만 접근할 수 있다. 기존 설계된 OpenGL SC 렌더링 엔진은 GPU를 사용하기 위해 GPU가 할당된 코어의 운영체제에서 동작하게 된다. 다음 Figure 6는 하나의 파티션에서 구동하는 렌더링 엔진 설계 개념도이다.

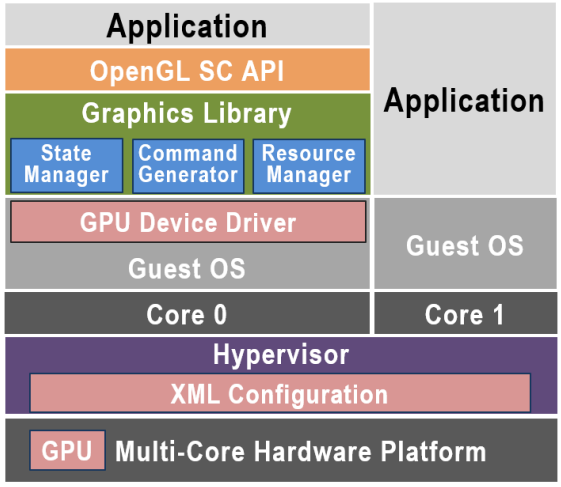

Fig. 6 OpenGL SC Rendering Engine Diagram in Single Partition

설계 개념도에서 보는 봐와 같이 Hypervisor의 XML Configuration을 통해 특정 코어에 GPU 자원이 할당되어 있으며, Guest OS 커널 영역에 GPU를 제어하기 위한 디바이스 드라이버가 탑재된다. 그래픽 렌더링 제어를 위한 그래픽스 라이브러리는 사용자 영역에서 제공되며, 그래픽스 라이브러리의 디바이스 드라이버 접근은 ioctl(Input/Output Control)을 사용한다.

그래픽스 라이브러리는 State Manger, Command Generator, Resource Manager로 구분된다. ‘State Manager’는 GPU 파이프라인 기능 설정에 대한 상태정보(OpenGL SC API를 통해 요청된 렌더링 정보)를 관리한다. ‘Command Generator’는 렌더링 요청 시 상태정보를 GPU의 파이프라인이 인식할 수 있는 명령어로 변환한다. ‘Resource Manger’는 GPU 메모리를 관리하며 렌더링 데이터(점, 선, 도형의 정점 데이터)를 GPU 메모리에 적재하는 기능을 가진다. OpenGL

SC API 레이어는 Khronos Group에서 제정한 국제표준 OpenGL SC 인터페이스를 제공한다.

Figure 7은 단일코어 처리환경에서 그래픽스 엔진의 분산처리를 위한 그래픽 렌더링 성능을 측정한 내용을 나타낸다. State Manager는 OpenGL SC API에 의해 가장 많은 변화가 발생함으로 'Application + OpenGL SC API + State manager'로 그룹화(Group A) 하였으며, 'Command Generator + Resource Manager' 그룹(Group B)와 'GPU Device Driver' 그룹(Group C)으로 구분하여 그래픽 렌더링 시 코어 사용량을 측정하였다.

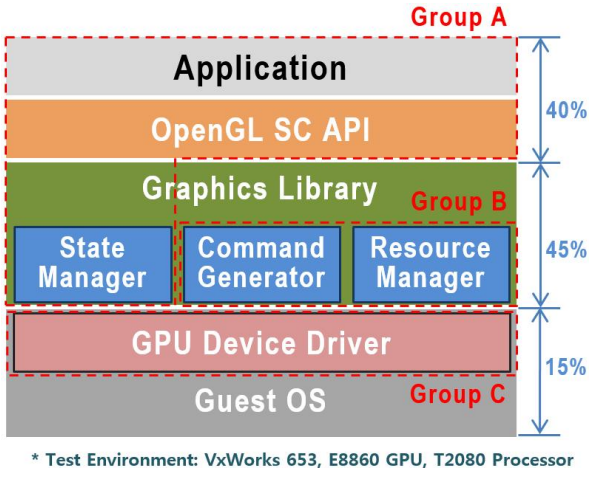

Fig. 7 Core Usage in Single Core Engine

항공기 시스템은 주어진 시간 내에 한 화면을 렌더링 해야 하며 이러한 주어진 시간을 운용주기라 한다. Fig. 7에서 보는바와 같이 운용주기 내에 각 그룹에서 사용한 코어 사용량을 보면 Group A와 Group B가 대부분의 시간을 소모하는 것을 볼 수 있다. 따라서 두 그룹을 독립된 파티션으로 분할하여 처리함으로써 성능 개선과 코어 점유율을 분산할 수 있다.

2.2.2 멀티코어 기반 렌더링 엔진 분산처리 설계

OpenGL SC API 중 정점 데이터를 사용하여 그래픽 렌더링을 요청하는 API(glDrawArrays, glEnd, glDrawElements)는 렌더링 완료를 위한 대기시간을 가지지 않는다. 단지 출력 장치에 화면을 갱신할 때에만 모든 렌더링이 완료될 때까지 대기할 뿐이다. 따라서 Fig. 7에서 구분한 Group A, B를 서로 다른 파티션으로 분산 처리할 경우 Group A와 Group B의 작업을 병렬로 처리할 수 있다.

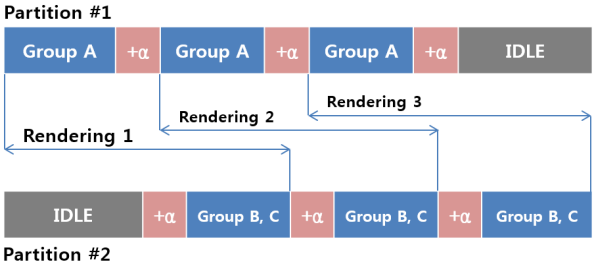

Fig. 8 Distributed Processing

이와 같은 분산처리를 위해서는 어플리케이션에서 요청한 렌더링 정보를 GPU에 전달하기 위한 파티션간 데이터 통신에 따른 부하가 발생한다. 따라서 전체적인 코어 사용량은 증가하게 된다. 그러나 렌더링 객체가 많을 경우 병렬처리에 따른 처리속도 향상 및 코어 점유율 분산 효과를 기대할 수 있다.

본 연구에서는 파티션간 렌더링 정보 공유를 위한 데이터 통신 부하를 최소화하기 위해 하이퍼바이저인 Module OS의 ARINC 653 서비스 중 Port를 이용하는 방법과 Shared Memory를 이용하여 설계하였다.

- *Port* : Guest OS는 하이퍼바이저에 의해 제공되는 채널을 할당받고 해당 채널을 통해서 필요한 데이터를 전송하거나 수신할 수 있다. VxWorks 653 3.x는 크게 Queuing Port와 Sampling Port를 제공하고 있다. Queuing Port의 경우 메시지 전달의 신뢰성에 높은 비중을 갖는 통신 방식으로 Queue 방식으로 데이터를 전송하거나 수신할 수 있으며, Sampling Port는 Queue 방식이 아닌 Overwrite 방식으로 갱신되는 형태로 데이터를 공유한다.

- *Shared Memory* : VxWorks 653 3.x는 Memory Configuration을 제공하여 Module OS상에 공유할 메모리를 할당한다. 할당된 메모리는 공유하고자 하는 파티션에 MMU를 통한 Shared Memory에 접근 가능 주소를 제공한다. Module OS는 파티션의 Shared Memory 접근에 관여하지 않는다. 그러므로 사용자는 파티션간 데이터 동기화에 대한 책임을 가진다.

Queuing Port는 통신에 있어 신뢰성이 높으나 Configuration 시점에 메시지 큐의 개수와 메시지의 최대 크기가 설정됨으로 통신 데이터가 가변이거나 메시지의 최대 크기를 넘어서는 경우 유용하지 않다. 그리고 메시지 큐 이벤트에 대해 Module OS가 관여함으로 Shared Memory를 사용하는 것에 비해 데이터 처리가 느리다. Shared Memory의 경우 데이터 공유가 빠르게 이뤄지나 파티션간 운용주기에 대한 동기화가 어렵다. 따라서 파티션간 운용주기 및 주요 동작에 대해서는 Queuing Port를 이용하여 제어하고, 데이터량이 많고 가변적인 그래픽스 렌더링에 필요한 데이터는 Shared Memory를 이용하여 운용한다. Figure 9는 그래픽스 렌더링 분산처리를 위해 Port 및 Shared Memory 서비스를 이용한 설계 개념도이다.

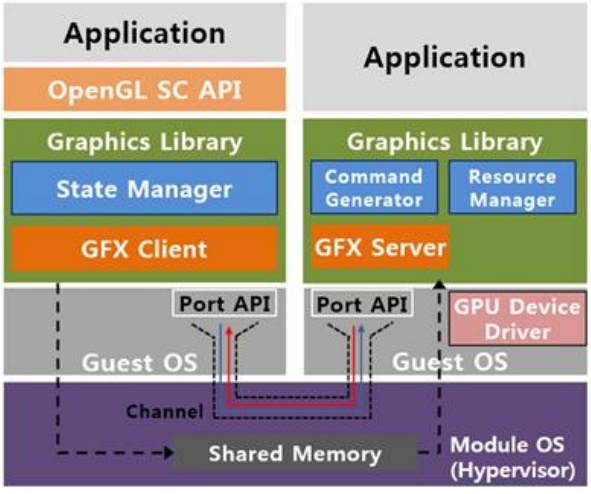

Fig. 9 OpenGL SC Multi-partition Rendering Engine Diagram

2.3 타깃 기반 시험 및 성능분석

2.3.1 시험환경 및 조건

설계된 멀티코어 기반 분산처리 그래픽스 렌더링 엔진의 성능을 측정하기 위해 Freescale의 4Core를 지원하는 T2080 Processing을 사용하였으며, 대화면 처리를 위해 ATI의 임베디드 GPU인 E8860을 사용하였다. 세부적인 주요 H/W 및 System 구성은 다음과 같다.

- Processor: T2080 4Core
- Graphics Processing Unit: E8860
- Resolution: 2560x1024
- Operating System: VxWorks 653 3.1.3
- Guest OS: VxWorks 6.6.8
- 운용주기: 40ms

Fig. 10 Graphics Rendering Test Environment

VxWorks 653 파티션 운용은 단일코어 렌더링 엔진과 분산처리를 위한 멀티코어 렌더링 엔진의 성능 비교를 위해 Figure 11과 같이 하나의 파티션을 사용한 단일 파티션 환경과, 2개의 파티션을 사용한 분산처리 환경으로 구성하였다.

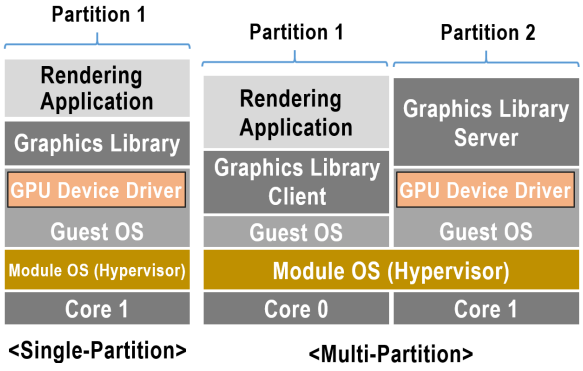

Fig. 11 Comparison Test Condition

그래픽스 렌더링 엔진을 분산 처리할 경우 Fig. 8에서 본바와 같이 파티션간 데이터 통신 부하가 발생하게 된다. 그러므로 시험을 위한 그래픽은 2D 그래픽을 렌더링에 있어서 가장 많은 파티션간 데이터 통신이 발생할 것으로 예상되는 심볼을 시험하도록 한다. 항

공기 계기 디스플레이에서 가장 많이 사용되는 심볼 형태는 Box, Texture, Circle을 예로들 수 있으며 파티션간 데이터 통신량은 다음과 같다.

- Box: (2Vertex + 4Color) * 4Byte
- Texture: (2Vertex + 2Texture Coordinate) * 4Byte
- Circle: (2Vertex + 4Color) * 360/Angle * 4Byte

서클은 라인을 일정한 단위로 회전하면서 반복적으로 그림으로써 생성할 수 있다. 그러므로 서클을 그리기 위해서 가장 많은 데이터 정보 생성과 통신 부하가 발생하게 된다. 따라서 테스트에 사용할 어플리케이션은 회전각을 2도 단위로 하는 서클을 렌더링하도록 구성하여 비교시험을 수행한다. 하나의 서클을 렌더링하기 위해 발생하는 파티션간 데이터 통신량은 다음과 같다.

(2Vertex * 4Color) * 360/2 * 4 = 4,320 Byte

2.3.2 시험결과 및 분석

시험 조건에 따라 비교시험 결과 서클을 100 ~ 500개를 렌더링하는데 사용된 CPU 사용량 비교측정 결과는 Figure 12과 같다.

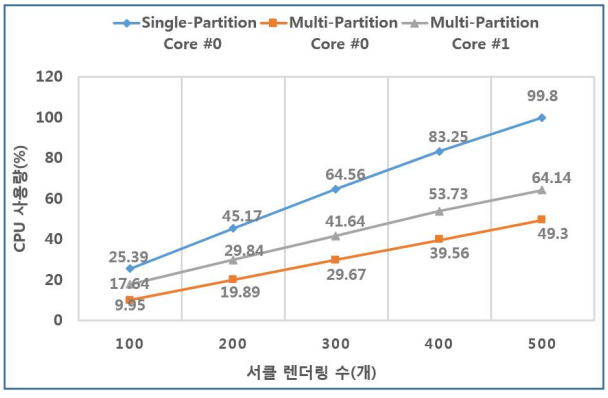

Fig. 12 Result of Cpu Usage Comparison Test

가로축은 운용주기 내에 렌더링한 서클의 수이며, 세로축은 운용주기 내에 주어진 서클을 렌더링하기 위해 사용한 CPU 사용량을 나타낸다.

Fig. 12의 결과를 보면, 단일 파티션 사용에 비해 멀티 파티션을 이용할 경우 CPU의 사용량이 분산되어 어플리케이션이 탑재된 파티션이 보다 많은 여유시간이 확보됨을 확인할 수 있었다. 어플리케이션에서 서클 500개를 렌더링할 경우에는 단일 파티션 운용환경은 운용주기를 넘어서는 현상이 발생하였으나, 멀티 파티션 운용환경에서는 운용주기 중 CPU를 약 50%를 사용하고 있는 것으로 나타났다. 이는 멀티 파티션으로 그래픽스 렌더링 엔진을 구성할 경우 단일 파티션 렌더링 엔진에 비해 운용주기 내에서 보다 많은 그래픽을 렌더링 할 수 있는 것으로 분석된다.

3. 결론

본 논문에서는 멀티코어 기반의 파티셔닝 운영체제에서 그래픽스 렌더링에 사용되는 CPU 사용량을 분산하여 시스템 운용주기 내에 보다 많은 그래픽을 렌더링할 수 있는 렌더링 엔진 분산처리에 대한 연구를 수행하였다. CPU 사용량을 분산하기 위해 렌더링 엔진을 2개의 파티션으로 분할하여 설계하는 구조를 제안하였으며, 분할에 따른 파티션간 데이터 공유는 CPU 사용량을 최소화하기 위해 ARINC 653 서비스인 Port와 Shared Memory를 사용하여 설계하였다. 설계된 분산처리 렌더링 엔진은 타깃 기반에서 테스트를 수행하여 유용성을 검증하였으며, CPU 사용량을 측정한 결과 단일 파티션에 비에 멀티 파티션 운용시 시스템 운용주기 내에 보다 많은 그래픽을 렌더링 할 수 있음을 확인하였다. 그러나 파티션 분할에 따른 통신부하에 의해 전체 코어에 대한 CPU 사용량이 증가하는 현상이 발생하였다. 향후에는 이러한 통신부하를 최소화하기 위한 방법에 대한 연구를 수행할 계획이다.

References

[1] Morgan, M.J., "Integrated modular avionics for next-generation commercial airplanes", *Proceedings of the IEEE 1991 National,* pp. 43-49 vol.1. NAECON, 1991.

[2] Wind River Systems, "ARINC 653 - An Avionics Standard for Safe, Partitioned Systems". *IEEE ±Seminal*. August 2008. Retrieved 2009-05-30.

[3] Khronos Group. Khronos group home page,

http://www.khronos.org/opengles/2_X.

[4] M. Segal and K. Akeley, The OpenGL Graphics System A Specification, version 4.1 (core profile), Khronos Group, 2010.

[5] A. Munshi and J. Leech, OpenGL ES Common Profile Specification, version 2.0.24(full specification), Khronos Group, 2009

[6] B. Stockwell , OpenGL|SC - Safety-Critical Profile Specification Version 1.0.1 (difference specification). March 12, 2009