J. Korean Soc. Aeronaut. Space Sci. 48(6), 467-478(2020)
DOI:https://doi.org/10.5139/JKSAS.2020.48.6.467
ISSN 1225-1348(print), 2287-6871(online)

위성비행소프트웨어를 위한 XtratuM 가상화 기반의 RTEMS SMP 플랫폼

김선욱[1], 최종욱[2], 정재엽[1], 유범수[3]

Development of RTEMS SMP Platform Based on XtratuM Virtualization Environment for Satellite Flight Software

Sun-wook Kim[1], Jong-Wook Choi[2], Jae-Yeop Jeong[1] and Bum-Soo Yoo[3]
Korea Aerospace Research Institute

ABSTRACT

Hypervisor virtualize hardware resources to utilize them more effectively. At the same time, hypervisor's characteristics of time and space partitioning improves reliability of flight software by reducing a complexity of the flight software. Korea Aerospace Research Institute chooses one of hypervisors for space, XtratuM, and examine its applicability to the flight software. XtratuM has strong points in performance improvement with high reliability. However, it does not support SMP. Therefore, it has limitation in using it with high performance applications including satellite altitude orbit control systems. This paper proposes RTEMS XM-SMP to support SMP with RTEMS, one of real time operating systems for space. Several components are added as hypercalls, and initialization processes are modified to use several processors with inter processors communication routines. In addition, all components related to processors are updated including context switch and interrupts. The effectiveness of the developed RTEMS XM-SMP is demonstrated with a GR740 board by executing SMP benchmark functions. Performance improvements are reviewed to check the effectiveness of SMP operations.

초 록

위성비행소프트웨어의 역할이 커짐에 따라 가상화 기술이 위성에도 도입되고 있다. 가상화 기술 중 하나인 하이퍼바이져는 하드웨어 자원의 가상화를 통해 하드웨어를 보다 효율적으로 쓸 수 있도록 도와준다. 동시에 가상화 기술은 소프트웨어의 복잡도를 낮추어 신뢰성을 높이는 역할도 수행한다. 한국항공우주연구원에서는 위성용 하이퍼바이져 중 하나인 XtratuM을 차세대 하이퍼바이져 후보군으로 선정하고, 이를 위성비행소프트웨어에 적용할 수 있는지 가능성을 확인하고 있다. XtratuM은 하드웨어 효율성을 높일 수 있지만 SMP를 지원하지 않아 인공위성의 자세제어 알고리즘과 같이 고성능/병렬처리가 필요한 부분에 적용이 어렵다는 한계점을 지니고 있다. 본 논문에서는 XtratuM의 기능 확장과 RTEMS XM-SMP BSP를 추가적으로 구현하여 RTEMS 기반 SMP를 지원하도록 만든다. XtratuM을 분석하여 SMP에 필요한 기능을 하이퍼콜로 추가한다. 그 후 BSP를 수정하여 SMP에 필요한 다수의 프로세서를 초기화하는 과정과 프로세서간의 통신을 위한 초기화 과정을 구현한다. 나아가 문맥 교환, 인터럽트와 같이 SMP에 의한 충돌이 발생할 수 있는 부분에 대해서도 개선한다. 이렇게 개발한 RTEMS XM-SMP는 4개의 코어를 가지고 있는 GR740 보드를 이용하여 SMP 벤치마크 함수를 수행하여 검증하고 SMP를 통한 성능 변화를 확인한다.

† Received : March 20, 2020 Revised : May 13, 2020 Accepted : May 19, 2020
[1,3] Senior Researcher, [2] Principal Researcher
[3] Corresponding author, E-mail : bsyoo@kari.re.kr

Key Words : Virtualization(가상화), XtratuM, RTEMS, Hypervisor(하이퍼바이저), Satellite Flight Software(위성비행소프트웨어), Symmetric Multi Processor(대칭형 멀티프로세서)

Ⅰ. 서 론

인공위성들은 다양한 종류의 탑재체를 가지고 5년 이상의 장기적인 기간 동안 지구 궤도상에서 주어진 임무를 수행한다. 나아가 인공위성들의 활동범위가 심우주(Deep space)까지 확장되면서 그 역할과 기능이 점점 커지고 있다. 이에 따라 인공위성을 제어하는 위성비행소프트웨어(Satellite flight software) 또한 점점 복잡해지고 있다.

위성비행소프트웨어의 복잡성은 안정성(Safety)과 신뢰성(Reliability)을 감소시키는 요인이다. 나아가 위성 전체의 안정성으로 이어지기 때문에 위성비행소프트웨어의 복잡도(Complexity) 증가에 따른 안정성과 신뢰성을 유지하는 방법이 필요하다. 그 중 한 가지 방법은 하이퍼바이저(Hypervisor) 기반 가상화(Virtualization) 기법이다. 하이퍼바이저는 시스템 자원을 가상화하고 독립적인 가상화 구동 환경인 파티션(Partition)을 다수 생성한다. 파티션은 시스템 자원에 접근할 수 없으며 하이퍼바이저를 통해서만 접근이 가능하다. 이러한 구조는 하이퍼바이저가 배타적으로 시스템 자원을 관리할 수 있게 만들어 공간적 분할(Time and space partitioning) 특징과 함께 하드웨어 사용의 효율성을 높인다. 위성비행소프트웨어의 각 부분들은 여러 파티션에 나눠져 동작하기 때문에 복잡도가 낮아지며 이는 위성의 안정성 및 신뢰성 향상으로 이어진다. 나아가 파티션 기반의 소프트웨어 구현은 소프트웨어 이식성을 향상시킴으로써 안정화된 소프트웨어의 재사용을 가능하게 한다.

하이퍼바이저 기반 가상화 기술을 이용하여 안정성과 신뢰성을 높이고자 하는 연구는 점점 늘어나고 있다[1]. 임베디드(Embedded) 시스템 분야에서는 KVM/QEMU/XEN 등의 상용 하이퍼바이저가 있다. 하지만 위성용 프로세서는 상용 프로세서와는 달리 우주환경에서도 정상적으로 동작하기 위한 엄격한 요구조건들이 있다[2]. 가상화 기술도 마찬가지로 상용 하이퍼바이저를 적용하기에는 한계가 있다. 위성용 프로세서에서 사용가능한 하이퍼바이저로는 AIR, XtratuM, PikeOS가 있다. AIR은 항공 분야에서 많이 쓰이는 하이퍼바이저로 ARINC653 표준을 바탕으로 구현되었다[3]. AIR는 모든 파티션들에게 동일한 운영체제의 서비스를 제공하여 마치 파티션이 운영체제의 프로세스처럼 동작하는 구조를 지닌다. XtratuM(XM)은 인공위성을 포함한 실시간 임베디드 시스템을 대상으로 하는 하이퍼바이저이다[4]. XM은 초기에 x86 구조를 목표로 설계되었지만 위성용 프로세서 중 하나인 SPARC LEON프로세서들을 지원하기 위해 재설계되었고 European Space Agency (ESA)의 인공위성에 사용되고 있다. PikeOS는 상용 하이퍼바이저임에도 불구하고 항공우주뿐만 아니라 국방, 교통, 의료, 가전을 포함한 안전성과 보안성이 중요한 분야에 주로 사용되고 있다[5].

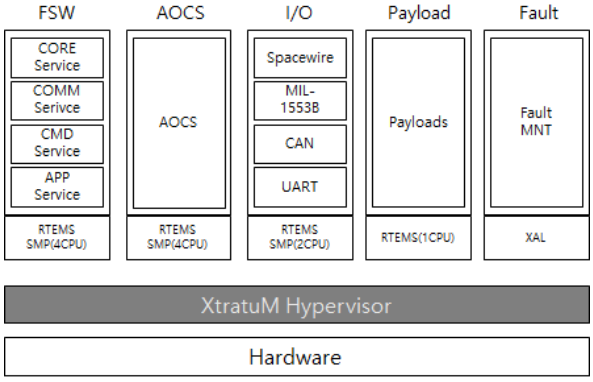

Fig. 1. Next generation satellite flight software

이러한 위성용 가상화 기술은 이미 다수의 우주프로젝트에 적용되어 안정성과 신뢰성 측면에서 검증된 상태이다. 하지만 기술적인 보안 문제로 인해 제한된 접근만 가능하며 이를 우리가 개발하는 인공위성에 적용하기 위한 기술지원이 거의 불가능하다. 따라서 세계 각국의 우주전문 기관들은 자신들의 비행소프트웨어에 적합하게 하이퍼바이저를 수정 및 개선하여 사용한다.

한국항공우주연구원에서는 차세대 인공위성에 가상화 기술을 도입하기 위하여 XM을 가상화 플랫폼 후보로 선정하고 위성비행소프트웨어에 적용할 수 있는지 그 가능성을 검토하고 있다. Fig. 1은 차세대 인공위성을 위한 가상화 기반의 위성비행소프트웨어의 구조이다. 총 5개 FSW, AOCS, I/O, payload, fault management 파티션으로 구성된다. 이 중에서 Flight Software(FSW)와 Attitude and Orbit Control System (AOCS)과 같이 고성능 프로세싱이 필요하거나 I/O와 같이 멀티 태스크가 필요한 파티션은 RTEMS 기반의 Symmetric Multi Processor(SMP) 프로세싱 환경을 사용하여 성능을 향상시킬 계획이다. RTEMS는 SPARC LEON 프로세서를 지원으로 하는 real time operating system(RTOS) 중 하나로 인공위성의 운영체제로 주로 사용되고 있으며, SMP 기반 멀티프로세싱 환경을 완벽하게 지원하고 있다. 하지만 XM 개발사인 FentISS에서 제공되는 RTEMS XM Board Support Package (BSP)는 단일 프로세서 환경만 지원하기 때문에 SMP 프로세싱을 이용한 고성능 병렬처리가 필요한 파티션은 지원이 불가능하다.

본 논문에서는 이러한 단점을 보완하여 XM 파티션 내에서 RTEMS SMP를 완벽하게 지원하는 RTEMS XM-SMP를 제안한다. XM을 분석하여 SMP에 필요한 기능들을 정리하고 하이퍼콜 형태로 추가한다. 그 후 단일코어(Singlecore) 기반의 XM BSP를 수정하여 SMP에 필요한 다수의 프로세서를 초기화하는 과정과 프로세서간의 통신을 위한 초기화 과정을 구현한다. 나아가 문맥 교환, 인터럽트와 같이 SMP에 의해 충돌이 발생하기 쉬운 부분에 대해서도 개선한다. 개발한 RTEMS XM-SMP는 인공위성용 프로세서인 LEON4 기반의 GR740 보드를 통해 검증한다. GR740 보드는 4개의 LEON4 프로세서를 탑재한 멀티코어(Multicore) 프로세서로 한국항공우주연구원에서 차세대 위성용 프로세서의 후보 중 하나로 선정하여 도입 가능성을 검토 중인 프로세서이다[6]. 사용한 벤치마크(Benchmark) 함수는 ParMiBench이며, 이를 이용하여 SMP 환경의 안정성과 SMP 환경에서의 성능 변화를 확인한다.

논문의 구성은 다음과 같다. 2장은 XM에 대한 소개 및 SMP 처리 구현을 위한 XM 기능 확장과 RTEMS XM-SMP BSP 개발에 대해 설명하고 3장에서 실험을 통한 구현된 시스템의 성능 결과에 대해 설명하며, 4장을 통해 결론을 맺는다.

II. 본 론

2.1 XtratuM 소개

XM은 반가상화(Para-virtualization) 방식의 하이퍼바이저로, 최근 ESA에서 개발되는 ARGOS, ANGEL, EyeSat, JUICE, PLATINO와 같은 다수 우주관련 프로젝트의 탑재컴퓨터에 적용되어 성능 및 안전성을 입증하였다[7].

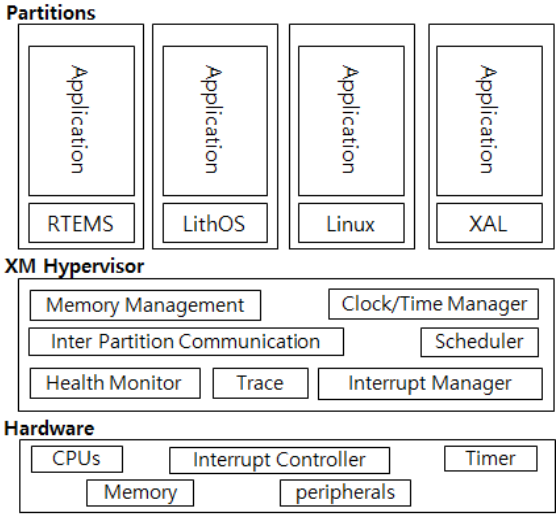

Fig. 2. Virtualized system based on XM

Figure 2는 XM을 이용하여 구축한 가상화 시스템이다. XM은 최하단의 하드웨어 자원을 정적 분할하고 가상화하여 가상의 CPU, 메모리, 인터럽트(Interrupt) 및 기타 하드웨어 자원을 생성한다. XM은 독립된 작업 공간인 파티션을 생성하고 가상화된 자원을 파티션에 제공한다. 파티션은 가상화된 자원을 제공받아 소프트웨어를 구동한다. XM은 이 모든 과정을 제어하며, 추가로 내부 파티션 간의 통신과 트레이스(Trace), 모니터 기능을 제공함으로써 구축된 가상화 환경의 신뢰성을 높인다. 파티션 내의 소프트웨어는 가상화된 공간에서 제공받은 하드웨어 자원만을 이용하여 동작하기 때문에 XM의 하드웨어 자원 관리는 신경 쓸 필요가 없다.

CPU의 가상화를 자세히 살펴보면, XM은 다수의 물리적인 CPU(Physical CPU; pCPU)를 이용하여 다수의 가상 CPU(Virtual CPU; vCPU)로 가상하고, 환경 구성을 통하여 각각의 파티션에 할당한다. 이 때 주기 스케줄러(Cycle scheduler)를 통해 시간적 분할을 수행하여 하나의 pCPU가 다수의 vCPU로 가상화될 수 있도록 만든다. 파티션에서는 오직 할당받은 vCPU만 이용할 수 있으며 어떤 pCPU가 자신의 파티션에 할당되었는지 인지할 필요가 없다. Fig. 3은 XM에서 XM 환경구성을 통한 CPU 할당 예시이다. vCPU는 같은 색으로 표시된 pCPU로부터 할당된 것을 나타낸다. XM은 4개의 pCPU를 가지고 총 10개의 vCPU를 생성하였다. 노란색으로 표시된 pCPU#2를 보면 4개의 파티션에 모두 할당되어 있다. 따라서 스케줄링 타이밍을 이용하여 가상화될 vCPU를 바꿔가면서 파티션#1의 vCPU#1, 파티션#2, #3, #4의 vCPU#2의 역할을 교대로 수행한다.

이와 같이 XM에서 하나의 pCPU는 가상화를 통해 다수의 vCPU역할을 수행할 수 있고, 다수의 vCPU들을 하나의 파티션에 할당할 수 있다. 하지만 현재 XM은 하나의 파티션에서 다수 vCPU들을 각각 별도의 목적으로 운용하는 Asymmeritic Multi-Processor

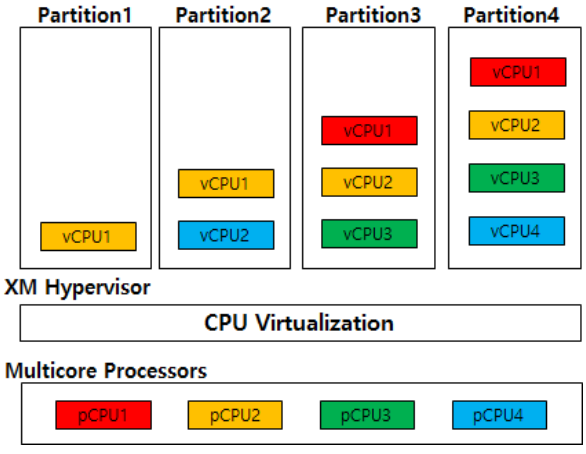

Fig. 3. Multi-Core allocation in XM

(AMP) 환경만을 지원하며, 여러 vCPU들을 동일한 목적으로 운용하는 SMP 환경은 고려하지 않고 있다. 또한 XM과 함께 개발사에서 제공하는 RTEMS XM BSP는 단일 vCPU를 위한 단일코어 RTEMS RTOS 만을 지원한다. 나아가 보안 문제로 더 이상 이용할 수 있는 자료가 없기 때문에 제조사에서 제공한 BSP를 우리의 인공위성에 맞게 직접 수정 및 개선해야 한다. 본 논문에서는 XM의 기능 확장과 RTEMS XM-SMP BSP 추가 구현을 통하여 XM의 파티션이 RTEMS 기반의 SMP 환경을 지원하도록 한다.

2.2 RTEMS XM-SMP를 위한 기능 분석

XM이 RTEMS SMP를 지원하기 위해서는 아래의 기능이 필요하다.

- 다수의 vCPU를 동일 시점에 하나의 파티션에 할당하는 기능
- SMP에서 사용하는 모든 vCPU에 대해 초기화하는 기능
- 내/외부에서 발생한 인터럽트를 특정 vCPU에 전달하는 기능
- RTEMS 스케줄링과 vCPU간의 통신을 위한 Inter Processor Interrupt(IPI) 기능

본 논문에서 사용하는 XM-4.6.1a는 AMP 환경을 지원하기 때문에 동일시점 다수 vCPU의 파티션 할당, 인터럽트 전달 기능을 가지고 있다. 하지만 SMP에서 사용하는 모든 vCPU를 초기화하는 기능과 파티션 내부의 vCPU간의 IPI 기능은 포함하지 않고 있다. RTEMS SMP를 XM 환경에서 구동하기 위해서는 XM 하이퍼바이저에 2가지 기능들에 대한 추가적인 구현이 필요하다.

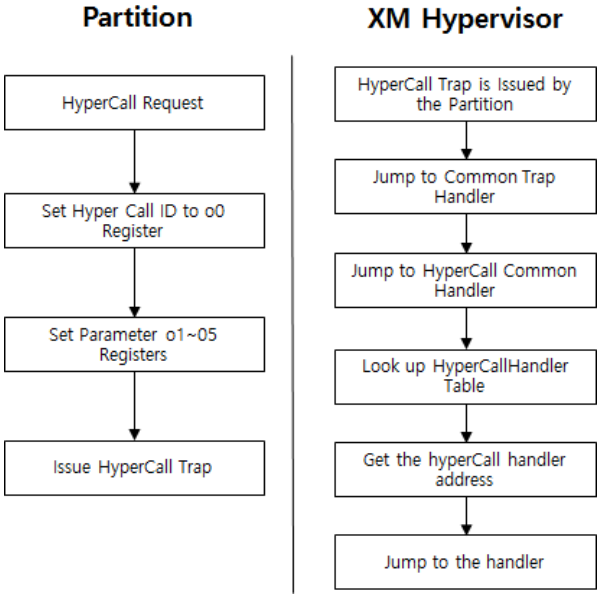

Fig. 4. XM HyperCall Processing

2.3 SMP를 위한 XtratuM 하이퍼콜

파티션 내부의 RTEMS은 하이퍼콜을 이용하여 XM에 하드웨어 자원 할당 및 서비스를 요청한다. 따라서 SMP를 위해 필요한 기능들은 하이퍼콜 형태로 구현하여 파티션 내부에서 호출이 가능하도록 만든다. XM 가상화 시스템에서의 하이퍼콜의 처리과정은 Fig. 4와 같다. 파티션에서 하이퍼콜을 요청할 경우 요청된 하이퍼콜의 번호를 LEON4 프로세서의 o0 레지스터(Register)에 저장하고, 요구되는 파티미터(Parameter)들을 o1~o5 레지스터에 저장한 후 LEON4의 ta (Trap always)명령을 사용하여 XM과 약속된 번호의 트랩(Trap)을 발생시킨다. XM의 모든 인터럽트는 XM 하이퍼바이저에서 처리함으로 해당 트랩은 XM 하이퍼바이저로 전달된다. XM 하이퍼바이저는 트랩 번호를 확인하여 하이퍼콜 요청임을 확인하고 o0레지스터를 통해 전달된 하이퍼콜 ID를 확인하여 해당 하이퍼콜의 핸들러(Handler)를 호출하여 처리한다.

본 논문에서는 RTEMS SMP를 지원하기 위해 Table 1의 2가지 하이퍼콜들을 XM 내부적으로 구현한다. XM의 하이퍼콜 프로세스에 적합하도록 하이퍼콜들에 대해 적절한 ID를 부여하고 이를 XM 내부의 하이퍼콜 테이블에 등록한다. 그러면 파티션에서 해당 하이퍼콜을 호출할 수 있다.

- int XM_get_vcpu_count(void)

SMP에서 사용하는 모든 vCPU에 대해 초기화를 수행하려면 파티션에 할당된 vCPU의 개수를 파악을 해야 한다. 따라서 파티션에 할당된 vCPU의 개수를 파악하는 하이퍼콜을 추가한다. 파티션 내부에 존재하는 vCPU의 개수는 XM 내부의 환경설정 정보를 참고하여 획득할 있다. vCPU의 개수 정보는 XM 환경 설정에 의해 파티션마다 가변적이기 때문에 XM 하이퍼바이저의 환경설정 정보를 통해 읽어온다. 이는 내부 프로그램과 vCPU의 개수를 분리시켜 내부 프로그램에서 vCPU의 개수에 대해 상관없이 구현이

Table 1. Hypercall added for SMP

HyperCall	Description
int XM_get_vcpu_count (void)	Returns the number of vCPU configured in the partition
int XM_gen_ipi (unsigned int target_cpu, unsigned int hwIrqMask, unsigned intextIrqMask)	Generates Inter Processor Interrupt between vCPUs configured in the partition

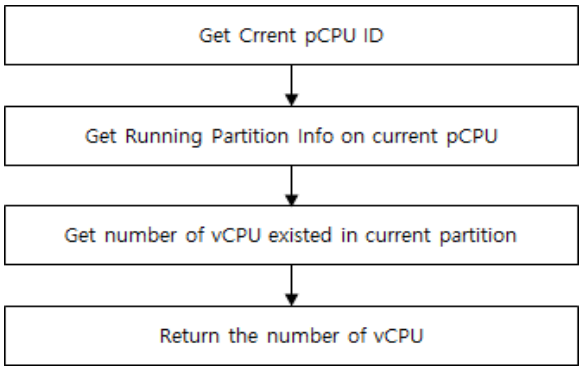

Fig. 5. XM_get_vcpu_count flow

가능하다. Fig. 5는 추가한 XM_get_vcpu_count() 하이퍼콜의 순서도이다. 하이퍼콜 발생 시 XM 내부에서 pCPU 단위로 관리되는 파티션 정보를 획득하고 이 정보에서 해당 파티션에 존재하는 vCPU의 개수를 구해 호출한 파티션으로 반환한다. 그에 따라 XM_ get_vcpu_count()의 입력 변수는 필요하지 않으며, 현재 파티션의 vCPU 개수를 반환한다.

- int XM_gen_ipi (unsigned int target_cpu, unsigned int hwIrqMask, unsigned int extIrqMask)

파티션 내부 vCPU간의 통신을 위한 하이퍼콜이다. IPI(Inter Processor Interrupt) 프로세싱은 파티션 내부에서 이뤄지는 과정이지만 인터럽트의 높은 우선순위, 수신할 vCPU의 상태, 그리고 파티션의 스케줄링 등을 고려할 때 반드시 하이퍼콜을 통한 내부 인터럽트로 처리하는 것이 바람직하다. XM의 특징을 고려하지 않고 파티션 내부 통신이라는 점에 집중하여 하이퍼콜이 아닌 인터럽트로 구현할 경우, 기존 인터럽트와의 충돌, 인터럽트간의 우선순위 반전(Inversion), 인터럽트를 요청한 vCPU와 수신하는 vCPU간의 스케줄링의 시간차와 같은 다양한 문제를 유발할 수 있으며 나아가 의도치 않은 파티션이나 XM 하이퍼바이저 상에서 인터럽트가 발생되는 문제가 발생할 수 있다.

Figure 6은 본 논문에서 구현한 파티션 내부 vCPU간의 IPI 처리과정이다. vCPU간의 IPI요청은 구현된 XM_gen_ipi() 하이퍼콜에 의해 이뤄지며 입력변수로 target_cpu, hwIRqMask, extIrqMask를 받는다. target_cpu는 어떤 CPU로 인터럽트를 전달할지를 나타내며 hwIrqMask는 해당 CPU에서 어떠한 인터럽트를 발생시킬지는 나타낸다. 이 값은 파티션 내부에서 사용되는 가상 인터럽트로 다른 파티션과 하드웨어와는 독립적이다. 마지막으로 extIrqMask는 XM의 tickless kernel 구조상 필요한 값으로 파티션에서 해당 인터럽트를 바로 수신하고 반응할 수 있게 도와준다. 연산 수행에 따라 성공 실패여부를 반환한다. XM_gen_ipi() 하이퍼콜이 발생하면 하이퍼콜 트랩에 의해 원본 vCPU를 가상화한 pCPU가 XM 하이퍼바이저의 하이퍼콜 핸들러로 분기한다. 하이퍼콜 핸들러에서는 IPI 파라미터들을 설정하고 대상 vCPU에 해당하는 pCPU의 인터럽트 컨트롤러를 이용하여 IRQ #12를 강제로 발생시켜 대상 pCPU가 XM하이퍼바이저의 IPI 핸들러로 분기하도록 한다. IRQ #12는 기능 확장을 위해 비워둔 인터럽트 번호로 사용가능하다. XM 하이퍼바이저로 분기한 대상 pCPU는 원본 pCPU에 의해 설정된 IPI 파라미터를 확인하고 원본 vCPU로부터 전달된 IPI 가상인터럽트(Virtual interrupt) 번호를 획득한다. 이후 pCPU에 해당하는 파티션에서 대상 vCPU의 가상인터럽트를 발생시킨다. 이렇게 발생된 가상 인터럽트는 파티션 내부의 vCPU로 전달되고 vCPU는 파티션 내부의 IPI 인터럽트 핸들러로 분기하여 해당 처리를 수행한다.

2.4 SMP를 위한 XtratuM BSP 구현

본 연구에서 사용되는 RTEMS-4.11.1 버전은 다양한 타겟 프로세서를 대상으로 SMP 구동환경을 지원한다. RTEMS SMP 모드로 구동하기 위해서는 BSP 측면에서 이와 관련된 요소의 구현이 필요하다. 하지

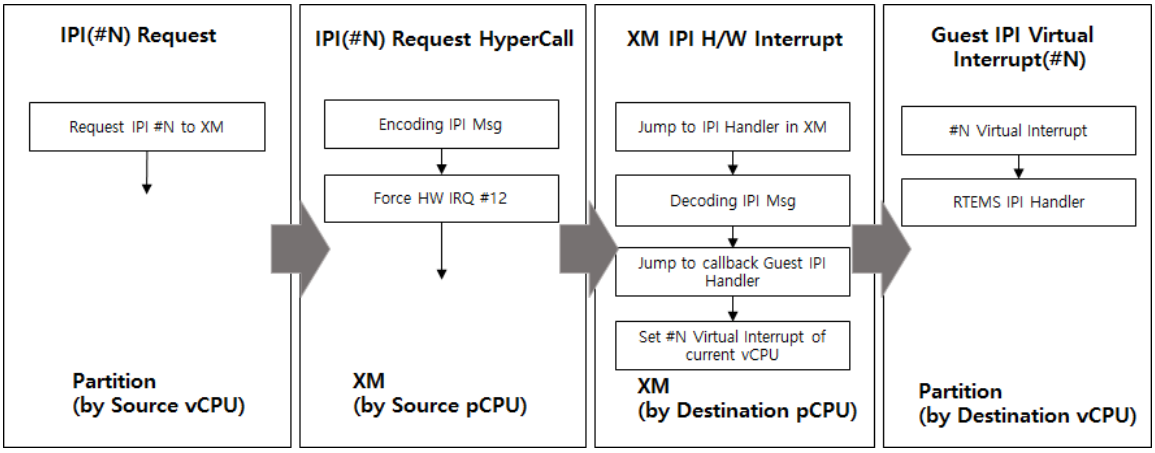

Fig. 6. Processing of the inter processor interrupt

만 XM이 제공하는 RTEMS 가상화 파티션을 위한 RTEMS BSP는 단일코어 환경만 지원하므로 RTEMS SMP를 완벽하게 지원하기 위해서는 RTEMS 커널(Kernel) 초기화, 스케줄링, 인터럽트와 같은 추가 기능이 필요하다.

XM 가상화 기반의 RTEMS XM-SMP BSP를 구현하기 위해서는 BSP 내부에서의 모든 CPU 제어가 vCPU를 기반으로 수행한다. Fig. 3에 나타낸 바와 같이 파티션에 할당되는 pCPU의 ID와 개수는 XM의 환경설정에 따라 유동적일 수 있고, pCPU의 물리적인 변경이 발생하더라도 파티션 내부에서는 이 변화에 대해 투명성 있게 동작되어야 때문에 RTEMS 내부에서는 반드시 vCPU를 사용하도록 구현되어야 한다.

- Get Current Processor CPU ID

RTEMS SMP 내부에서는 커널 내부의 스케줄링, 인터럽트 등 RTOS의 운용을 위해 현재 CPU의 ID를 확인하는 작업이 필요하다. 이를 위해 BSP 내부에 커널에서 현재의 CPU ID를 확인하는 함수인 _CPU_SMP_Get_current_processor()를 구현한다. 본 논문에서는 XM 가상화 환경의 RTEMS는 vCPU 단위로 운용되므로 XM_get_vcpuid()라는 XM 하이퍼콜을 이용하여 현재 vCPU의 ID를 반환하도록 구현한다.

- Startup Code

XM 가상화 시스템에서 RTEMS SMP 파티션이 시작되면 파티션 내에 할당된 vCPU들 중 vCPU0가 RTEMS의 엔트리 함수인 startup()으로 분기한다. Fig. 7은 SMP 환경에서의 startup() 흐름도를 나타낸다. 먼저 현재 수행중인 vCPU의 ID를 판단하여 ID가 0일 경우 RTEMS의 커널의 초기화를 진행하고 RTEMS SMP에서 사용하고자 하는 다른 vCPU들을 Wakeup하는 절차를 통해 모든 vCPU들이 RTEMS 환경으로 진입하도록 구현한다. startup과정에서는 XM의 하이퍼콜인 XM_get_vcpuid()를 사용하여 vCPU의 ID를 확인해야 하며, Wakeup된 모든 vCPU들은 XM 내부 라이브러리 함수인 init_ libxm()을 호출하여 XM으로부터 파티션 제어 정보에 대한 포인터들을 각 vCPU 자료 구조에 등록한다. 여기서 등록된 정보들은 파티션에서 사용하는 가상메모리 매핑정보와 인터럽트, IO포트 등으로 XM 하이퍼바이져 내부에서 파티션 스케줄링에 의한 문맥 및 인터럽트 처리를 위해 사용한다. 이러한 방식은 vCPU 생성 후 init_libxm()을 호출하는 것에 비해 확장성(Scalability)을 떨어뜨린다. 하지만 프로그램의 신뢰성을 높이기 위해 개발사의 방식을 그대로 유지했다.

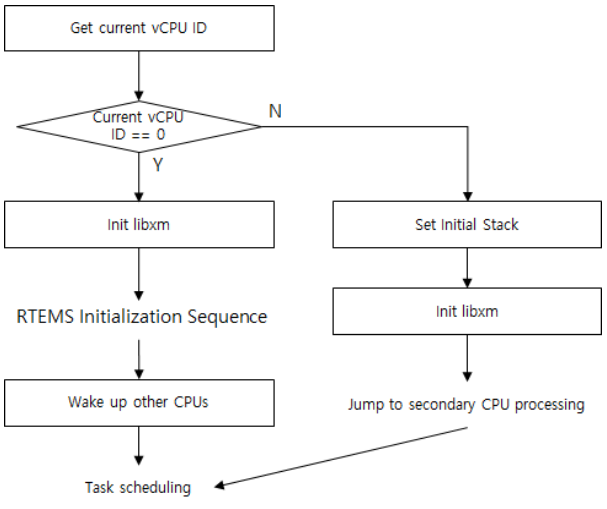

Fig. 7. RTEMS XM SMP Startup flow

- Wakeup other CPUs and join to scheduler

비가상화 기반의 멀티코어 환경에서는 시스템의 리셋(Reset)이 인가되면 하나의 CPU가 Wakeup되고, 나머지 CPU들은 그에 의해 커널의 초기화가 완료된 후 하드웨어로 구현된 인터럽트, 파워, 리셋 제어를 이용해 다른 CPU들을 하드웨어 적으로 Wakeup 시킨다.

XM 기반의 환경에서는 비가상화 멀티코어 환경과 유사하게 XM 하이퍼바이져가 모든 pCPU들을 물리적으로 준비시키고 관리한다. Fig. 8은 XM 파티션 vCPU의 유한 상태 기계(Finite state machine)를 나타낸다. vCPU들은 Boot, Running, Suspend, Halt Status 중의 하나의 상태를 가진다. XM 부팅 시에는 모든 vCPU가 Halt 상태를 갖게 되며, XM 초기화가 끝나고 파티션이 초기화될 때 해당 파티션에 할당된 vCPU0만이 리셋 명령에 의해 Boot로 전환되고 XM 내부에서의 CPU 자료구조의 할당, 스택(Stack), 메모리관리(Memory Management Unit) 등의 초기화가 끝나면 Normal 상태로 전환되어 파티션 내부의 프로세싱을 시작하게 된다. 이후 하이퍼콜에 의해 저전력 모드인 Suspend 모드나 파티션 종료나 오류 처리를 위한 Halt 모드로 전환될 수 있다.

XM의 파티션 스케줄링에 의해 파티션이 시작하면 파티션에 할당된 vCPU 중 vCPU0만이 Boot 모드로 전환되어 초기화 되고 Running 모드로 전환되어 파티션 코드를 시작한다. 나머지 vCPU들은 논리적인 Halt 모드로 vCPU0에 의해 Boot Mode로 전환될 때까지 XM 하이퍼바이져 내부에서 대기한다.

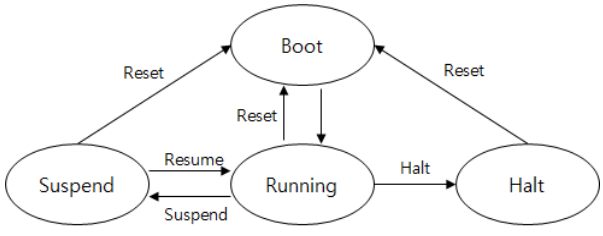

Fig. 8. Finite state machine for vCPU in XM

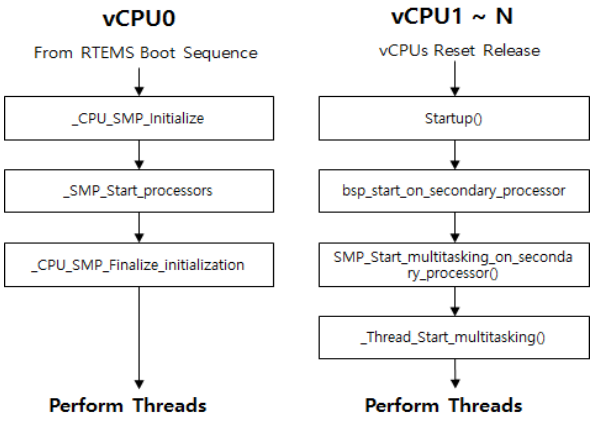

Fig. 9. vCPU Wakeup flow

Figure 9는 XM 가상화 환경에서 vCPU0와 나머지 vCPU들의 Wakeup 알고리즘을 나타낸다. vCPU0는 RTEMS 커널의 초기화를 완료한 후 첫 번째로 BSP에 구현된 _CPU_SMP_Initialize() 함수를 이용해 vCPU들을 Wakeup 시키기 위한 준비를 한다. 여기서 XM_get_vcpu_count() 하이퍼콜을 사용하여 파티션 내부에 할당된 vCPU의 개수를 구하며, vCPU의 개수를 이용하여 커널 내부의 CPU제어 자료구조를 초기화 하고, 다음 단계인 _CPU_SMP_Start_processor()에서 활용할 수 있도록 만든다. _CPU_SMP_Start_processor()는 실제 vCPU들을 Wakeup 시키는 함수로 앞서 구한 vCPU의 개수만큼 vCPU들의 리셋을 해제시키도록 구현한다. vCPU리셋을 해제하기 위해 XM의 하이퍼콜인 XM_reset_vcpu()를 이용한다. 마지막으로 _CPU_SMP_finalize_initialization()은 vCPU들을 Wakeup 시킨 후 추가적으로 필요한 절차를 수행하는 함수로 확장성을 위해 만들어둔다.

이와 같은 절차에 의해 vCPU0는 나머지 vCPU들을 Wakeup 시키게 되며, Wakeup 된 vCPU들은 RTEMS 엔트리 함수인 startup부터 수행한다. 이들은 startup 수행 후, vCPU0와 달리 bsp_start_on_secondary_processor()를 수행한다. 그 후, SMP_Start_multitasking_on_secondary_processor()와 _Thread_start_multiprocessing()을 통해 스케줄러를 호출한 후 쓰레드(Thread)를 수행한다.

- RTEMS IPI Processing

RTEMS SMP는 스케줄링과 동기화를 위해 BSP에 IPI 발생 함수인 _CPU_SMP_Send_interrupt()에 대한 구현이 필요하다. XM_gen_ipi() 하이퍼콜을 이용하여 RTEMS 파티션 내부의 vCPU간에 IPI 기능을 수행한다. XM에서는 각 CPU마다 할당된 인터럽트 컨트롤러를 사용하여 특정 CPU에 IPI를 위해 지정된 인터럽트를 발생시키며 인터럽트를 받은 대상 CPU는 IPI 핸들러를 이용하여 정해진 서비스를 수행한다. 본 논문에서는 RTEMS의 IPI를 위해 사용하지 않는 인터럽트 번호인 #15를 IPI 인터럽트로 할당한다.

- Context switch code patch

RTEMS는 멀티 쓰레드(Multi Thread)를 지원하는 운영체제로 쓰레드의 선점(Preemption), 동기화(Synchronization) 등에 의해 문맥전환(Context Switching)이 발생할 경우 수행 중이던 쓰레드의 문맥을 쓰레드 제어 블록(Thread Control Block; TCB)에 저장하는 문맥저장(Context Backup) 과정을 수행하고, 다른 쓰레드의 문맥을 로딩(Context Restore)하여 쓰레드를 수행한다. 단일코어 환경에서는 이러한 문맥교환이 발생할 경우 오직 하나의 CPU만이 그 절차를 수행하기 때문에 문맥저장과 새로운 쓰레드의 문맥로딩 과정에서 고려할 것이 많지 않다. 하지만 다수의 CPU가 다수의 쓰레드를 동시에 구동하는 SMP 환경에서는 짧은 시간에 여러 CPU에서 문맥교환이 다수 발생할 수 있기 때문에 CPU간의 동기화 문제가 고려되어야 한다. 특히 CPU에서 구동 중이던 쓰레드가 Swap Out 될 때 문맥저장 절차가 완전히 끝나기 전에 다른 CPU에 의해 해당 쓰레드가 Swap In 되거나, Swap In 과정에서 다수의 CPU가 동일한 쓰레드의 문맥로딩을 위한 경쟁상태(Race Condition)가 발생할 수 있다. 따라서 RTEMS BSP의 문맥교환 함수인 _CPU_Context_switch()와 _CPU_Context_restore()에서는 CPU간의 동기화를 고려하여 구현해야 한다.

본 논문에서는 멀티코어 환경의 문맥전환 과정에서의 동기화 문제를 해결하기 위하여 RTEMS Context_Control 자료구조에 존재하는 Executing_Flag를 이용하였다. Fig. 10은 본 연구에서 구현한 쓰레드의 문맥저장과 문맥로딩 과정을 나타낸 것이다. 특정 vCPU에서 문맥교환이 발생할 경우 기존 쓰레드에 대한 문맥저장이 완료되면 Executing_Flag를 0으로 설정하여 다른 vCPU에게 기존 쓰레드의 문맥저장이 완료되었음을 표시한다. 이후 해당 vCPU는 새로운 쓰레드의 문맥로딩을 시도하게 되는데 이때 다수의 vCPU가 동시에 동일한 문맥로딩을 시도하는 경쟁상태가 발생할 수 있다. 이를 해결하기 위하여 SPARC 프로세서

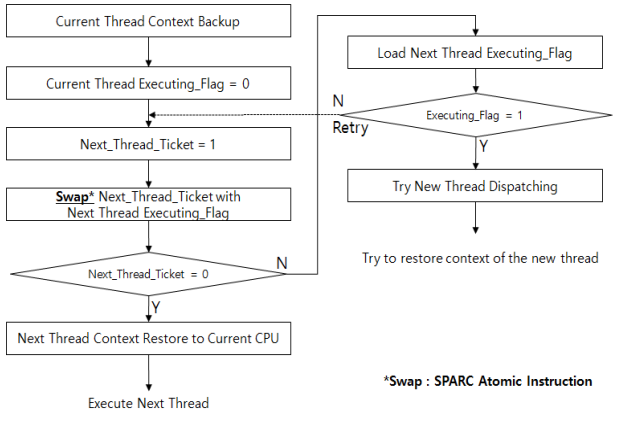

Fig. 10. Context switch code patch

에서 제공하는 원자명령(Atomic Instruction)인 swap를 사용하였다. 멀티코어 환경에서 코드상 동일한 위치의 swap 명령이 여러 CPU에서 동시에 시도될 경우 SPARC 프로세서는 오직 하나의 CPU만 명령을 성공하게 한다. 여러 vCPU들은 Next_Thread_ Ticket을 1로 설정한 후 0의 값을 갖는 Executing_ Flag와 swap를 시도하게 되는데 이때 swap 명령에 성공한 하나의 vCPU만 Next_Thread_Ticket과 Executing_Flag 값을 교환에 성공하여 Next_Thread_Ticket 값이 0으로, Executing_Flag 값이 1로 변경된다. 이때 swap에 실패한 나머지 vCPU들은 Next_Thread_Ticket 값을 1로 유지한다. swap에 성공한 vCPU는 새로운 쓰레드의 문맥을 로딩하여 수행하고, swap에 실패한 vCPU들은 동일한 쓰레드의 Executing_Flag 값이 0인지 확인하고, 0일 경우 해당 쓰레드의 문맥로딩을 다시 시도하게 되고, 그렇지 않을 경우 해당 쓰레드가 이미 다른 vCPU에서 실행중인 것으로 판단하고 대기 중인 다른 쓰레드의 로딩을 시도하게 된다. 만약 대기중인 쓰레드가 없을 경우 Idle 쓰레드를 수행한다.

- IPI를 위한 인터럽트 할당

인터럽트 측면에서 SMP환경은 단일코어 환경과 달리 인터럽트에 따라 그 전달되는 대상 CPU가 다르다. 본 논문에서는 Table 2와 같은 테이블을 사용하여 인터럽트마다의 대상 vCPU들을 관리하고 운용하도록 구현한다.

III. 성능분석

본 논문에서는 RTEMS SMP이 안정적으로 동작하고 SMP 프로세싱을 통해 성능향상에 기여를 할 수 있는지 확인한다. 성능의 측정은 멀티코어 임베디드 시스템을 위한 오픈소스 벤치마크 세트인 ParMiBench를 사용하여 진행하였다[8]. ParMiBench는 다양한 분야의 임베디드시스템 성능분석을 위해 널리 사용되어지는 MiBench를 병렬화하여 멀티코어 시스템의 성능분석에 적합하도록 변경한 것이다[9]. ParMiBench도 MiBench와 마찬가지로 임베디드 분야의 다양성에 기반한 다양한 형태의 알고리즘(Algorithm)을 벤치마크 함수로 제공한다.

Table 2. Interrupt target vCPU

IRQ Number	Target vCPU
17	0
18	1
19	0
20	3
21	2
22	0
.....	.....

Table 3. Hardware experiment environment

Factor	Specification
CPU Architecture	LEON4
Operating Frequency	250 Mhz
Core Number	4
Floating point unit	GRFPU
L1 instruction Cache	16 KiB
L1 data Cache	16 KiB
L2 Cache	Enabled
SDRAM	128 MiB

Table 4. Software experiment environment

Factor	Specification
XtratuM Version	4.6.1a
RTEMS	4.11.1
Build Optimize Option	-O2
Floating point unit	-mhard-float

3.1 실험환경

Table 3은 GR740 프로세서를 기반으로 제작된 보드의 하드웨어를 나타낸다. GR740 프로세서는 4개의 LEON4 프로세서를 가지며 각각 16KiB의 L1 Instruction/Data 캐시와 2MiB의 L2 캐시를 가지고 있다. L2캐시도 내장되어 있으나 본 논문에서는 사용하지 않는다. 또한 소프트웨어 구동을 위한 128MiB의 SDRAM은 가상화 환경 구동에 충분하다.

Table 4는 본 실험에 사용된 소프트웨어 환경을 나타낸다. XM은 최신 버전인 4.6.1a 버전을 사용하였으며 RTEMS는 SMP의 지원을 위해 4.11.1 버전을 사용했다. RTEMS 빌드 최적화 레벨은 2를 적용하기 위해 O2옵션으로 빌드하고, 하드웨어 부동소수점 처리장치를 사용하도록 빌드하여 성능측정을 진행했다.

Table 5는 본 연구에서 사용한 ParMiBench 알고리즘들과 그에 따른 파라미터 설정을 나타낸다. ParMiBench에서 제공하는 모든 분야의 벤치마크의 20개 알고리즘을 사용하여 성능을 측정하였으며 입력데이터의 크기는 시스템의 메모리를 고려하여 결정했다. 테스트의 반복횟수는 데이터로딩과 알고리즘 초기화 과정에 의한 영향을 최소화 하도록 큰 값으로 설정했다.

Table 5. ParMiBench Algorithm and Parameter

Test Category		Test Algorithm	Parameters
Automotive	BitCount	bitcnts bitcnts_1 bitstring	92,000 Iteration 112,500 Iteration 18,000 String length, 500 Iteration
	Susan	smooding edge corner	2,000 x 1,490 pixel image
	BasicMath	Solve Cubic Equations Calculate Integer Square Roots Calculate Long Square Root Perform Degree to Radian Angle Conversion Perform Radian to Degree Angle Conversion	Small Data Set (500 Mega Number)
Network	Dijkstra	Single Queue Multi Queue All Path	2,000 Nodes 2,000 Nodes 160 Nodes
	Patricia Trie	Patricia Trie	5,000 Nodes
Office	String Search	Pratt-Boyer-Moore Case-sensitive Boyer-Moore-Horspool Case-Insensitive Boyer-Moore-Horspool Boyer-Moore-Horspool	3 MiB Data pool, 40 Pattern search
Security	SHA	SHA-1	300 KiB x 13Set

3.2 성능측정결과 분석

성능의 측정은 XM의 가상화 환경에서 RTEMS SMP 파티션에 할당되는 vCPU들의 개수를 1개에서 4개까지 가변하며 ParMiBench 함수들의 성능을 측정하였다. 성능의 측정은 XM에서 제공하는 하드웨어 타이머 하이퍼콜을 이용하여 측정하였으며 측정된 벤치마크 시간을 바탕으로 성능향상 비율(Speed Up)과 코어들의 사용 효율(Efficiency)을 계산하여 본 논문에서 개발된 XM 기반의 가상화 환경에서 RTEMS SMP 가상화 환경에 대한 효용성을 분석했다.

성능향상 비율은 식 (1)에 의해 단일코어 성능결과(Ts) 대비 vCPU의 개수 증가에 따른 성능결과(Tp)의 성능향상비율(Speed Up)을 계산하여 분석하였다.

$$Speed\ Up = \frac{T_M}{T_S} \quad (1)$$

Figure 11은 모든 벤치마크 알고리즘에 대해 성능향상 비율을 측정한 결과이다. Y축의 성능을 나타내며 X축은 core의 개수를 나타낸다. Y의 성능은 단일코어의 성능으로 표준화하여 1.00이 단일코어일 때의 성능을 나타낸다. 대부분의 벤치마크에서 vCPU의 개수에 비례하여 선형적인 성능향상을 나타냈다. 하지만 bitstring, Dijkstra Single Queue(squeue), Patricia Trie에서는 vCPU의 개수가 일정 수 이상 증가할 경우 오히려 성능이 감소하는 것으로 나타났다. 이는 vCPU 증가와 벤치마크 내부의 태스크의 개수가 증가에 따라 vCPU와 태스크 간의 자료구조 접근에 대한 경쟁상태(Race condition)에 따른 동기화 오버헤드(Overhead)와 태스크와 vCPU 수의 증가에 따른 스케줄링 오버헤드의 증가 때문이다. XM 가상화 환경에서는 스케줄링 과정에서 XM 하이퍼바이져를 거쳐가기 때문에 비가상화 환경에 대비하여 스케줄링 오버헤드가 증가하였다. Table 6은 태스크의 동기화 및 RTEMS 스케줄링에서 사용되는 CPU 내부의 컨트롤 레지스터 접근시간을 측정 비교한 것이다. 실제 비가상화 환경에서 CPU는 내부 레지스터에 접근하기 위하여 1 사이클이 소요되는데 비해 XM 환경에서 vCPU에서 컨트롤 레지스터들의 접근들은 모두 하이퍼콜을 통해서 이뤄지므로 비가상화 환경보다 더 큰 오버헤드를 발생시키게 된다. 이러한 오버헤드는 vCPU의 개수가 증가하고, 그 횟수가 빈번할수록 증가하게 되기 때문에 응용프로그램 작성에 주의해야 한다.

실제 이러한 가상화 오버헤드의 영향을 측정하고자 성능이 감소한 3가지 알고리즘을 대상으로 XM 가상화 환경의 RTEMS SMP가 아닌 비가상화 환경의 RTEMS SMP에서 벤치마크를 측정하였다. Fig. 12에 나타난 바와 같이 bitstring과 Dijkstra Single Queue (squeue)에서는 가상화 환경에 비해 스케줄링 오버헤드가 감소하기 때문에 CPU 개수가 4까지 증가하더라도 미세한 성능 향상을 보이는 것으로 확인되었다. 하지만 Patricia Trie의 경우 여전히 CPU 개수가 3개 이상일 경우 오히려 그보다 적을 때보다 성능이 감소되는 것으로 나타났다. 이는 Patricia Trie 벤치마크 알고리즘의 특성상 Trie 생성과정에서 병렬화가 불가능하기 때문에 CPU 코어 증가에 비례한 병렬화 효과를 나타내기 어렵기 때문이다[10].

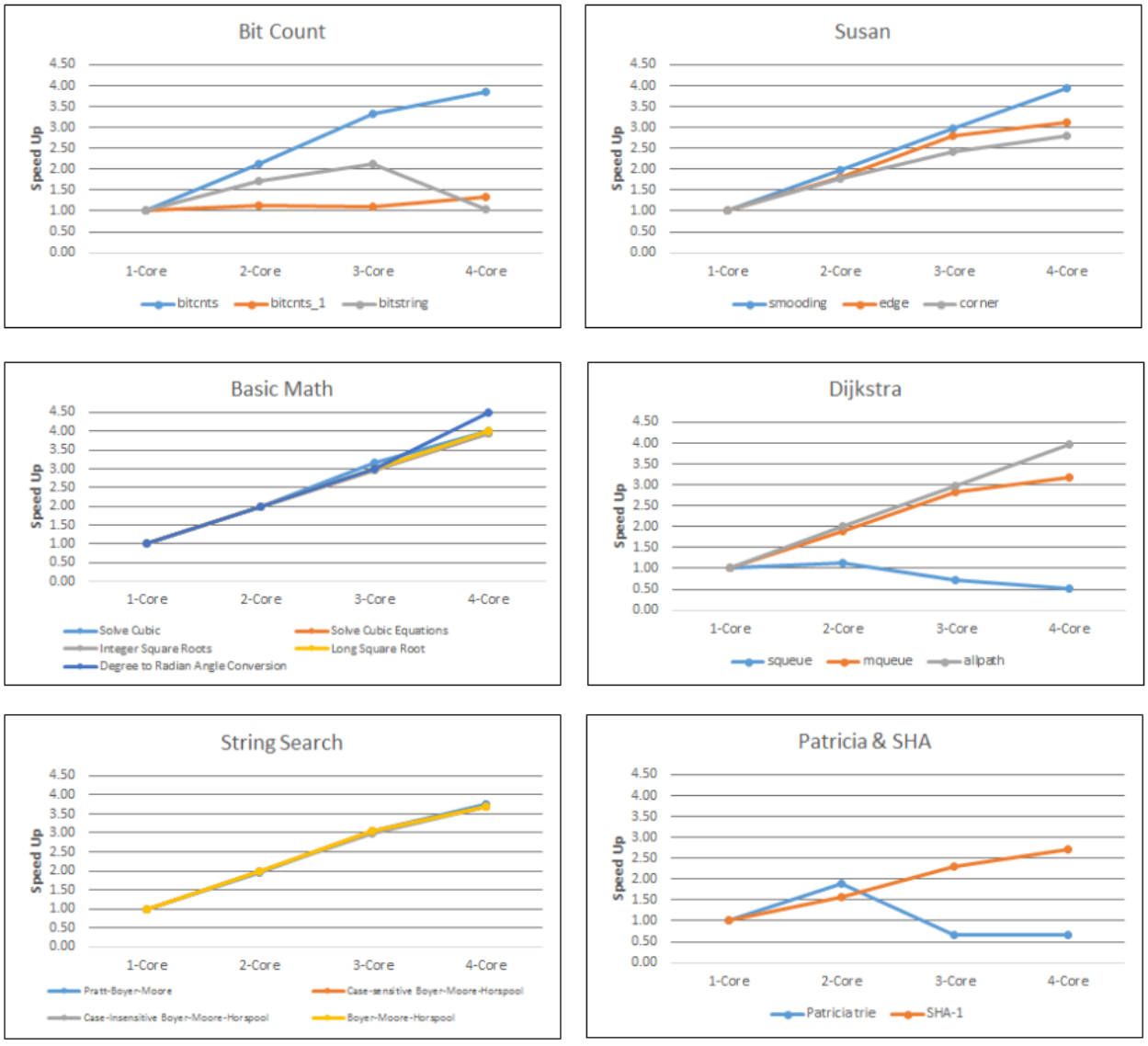

Fig. 11. Performance of the RTEMS XM-SMP based on XtratuM virtualization

Table 6. CPU control register access time

Function	RTEMS SMP	RTEMS XM-SMP
Read Process Status Register	4 ns	112 ns
Write Process Status Register	4 ns	392 ns
Set Process Interrupt Level	4 ns	56 ns
Clear Process Interrupt Level	4 ns	304 ns
Flush Register Window	4 ns	184 ns
Process Register Window Overflow	4 ns	184 ns

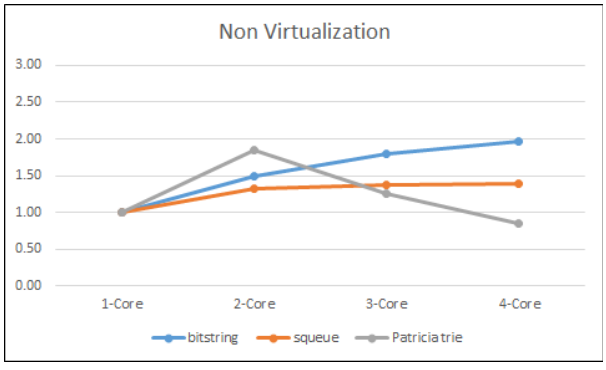

Fig. 12. Non-Virtualized performance

위 세 가지의 벤치마크를 제외한 대부분의 벤치마크에서는 vCPU의 개수에 따라 선형적인 성능향상 비율을 보였다.

$$Efficiency = \frac{Speed\ Up}{N_P} \quad (2)$$

Table 7은 성능향상비율에 식 (2)를 이용하여 vCPU의 개수(Np)에 증가에 따른 벤치마크별 효율성을 계산한 것이다. 각각의 값은 단일코어일 때로 표준화하였기 때문에 1보다 크거나 가까울수록 효율이 좋은 것을나타내고 1보다 작을수록 vCPU 개수 대비 효율이 떨어지는 것을 나타낸다. 효율성 수치는 알고리즘의 병렬화 정도를 나타내는 동시에 해당 시스템에 SMP 도입 여부를 판단하거나 최적화된 vCPU 개수를 선택하는데 활용될 수 있다. 가상화 플랫폼은 하나의 시스템에 다수의 가상 플랫폼이 존재하기 때문에 시스템 최적화를 위해 각각의 가상시스템마다 적정수의 vCPU를 분배하는 과정이 필요하다. 이러한 효율성 분석은 가상시스템의 특성에 따른 vCPU 개수를 결정하는데 도움이 될 수 있다.

본 논문에서는 개발한 XM 가상화 시스템 기반의 RTEMS XM-SMP 환경에서 ParMiBench를 이용하여 안정성 및 성능을 확인하였다. 대부분의 벤치마크에서 vCPU의 개수에 비례한 선형적인 성능향상과 효율성을 나타냈다. 몇몇 벤치마크는 할당된 vCPU갯수에 비해 낮은 성능의 결과를 나타냈는데 XM하이퍼바이저의 오버헤드와 벤치마크 알고리즘 특성에 의한 것으로 가상화 플랫폼이 제공하는 구조적인 효율성을 고려할 때 충분히 납득할만한 성능저하라 판단한다. 성능 결과로 미뤄볼 때 가상화 환경에서의 SMP 시스템의 성능은 OS와 하이퍼바이저의 개선을 통한 SMP 구동환경의 제공도 중요하지만 무조건 적인 SMP 플랫폼의 적용보다는 해당 가상화 파티션의 응용프로그램의 특성을 통해 적합한 vCPU의 개수를 선택하는 절차도 선행되어야 할 것이다. 그렇지 않을 경우 효율성이 저하되어 기대 이하의 성능을 나타내거나 오히려 SMP 적용 후 성능이 저하될 수 있다.

Table 7. Multi-Core efficiency

Test Algorithm \ Number of Core	1	2	3	4
bitcnts	1.00	1.06	1.11	0.96
bitcnts_1	1.00	0.57	0.37	0.33
bitstring	1.00	0.86	0.71	0.26
smooding	1.00	1.00	0.99	0.99
edge	1.00	0.90	0.94	0.78
corner	1.00	0.89	0.80	0.70
Cubic Equations	1.00	1.00	1.05	1.00
Integer Square Roots	1.00	0.99	0.99	0.99
Long Square Root	1.00	0.99	0.99	0.99
Degree to Radian Angle Conversion	1.00	1.00	1.00	1.00
Radian to Degree Angle Conversion	1.00	1.00	1.00	1.12
squeue	1.00	0.57	0.24	0.13
mqueue	1.00	0.95	0.94	0.80
allpath	1.00	1.00	1.00	0.99
Patricia trie	1.00	0.95	0.22	0.16
Pratt-Boyer-Moore	1.00	0.98	1.02	0.94
Case sensitive Boyer-Moore-Horspool	1.00	0.99	1.01	0.92
Case Insensitive Boyer-Moore-Horspool	1.00	0.99	1.00	0.92
Boyer-Moore-Horspool	1.00	0.99	1.01	0.92
SHA-1	1.00	0.78	0.77	0.68

Ⅳ. 결 론

본 논문에서는 인공위성분야의 가상화 환경 구축에 주로 사용되어지는 XM하이퍼바이저와 RTEMS RTOS를 기반으로 인공위성을 위한 고성능 가상화 시스템 구축을 위해 가상 vCPU기반의 SMP 환경을 구축하였다. 이를 위해 XM하이퍼바이저의 기능 확장과 RTEMS XM-SMP BSP를 추가 구현하고 벤치마크 세트를 이용하여 구현된 vCPU의 개수를 가변화 하면서 가상화 시스템의 성능을 분석하였다. 구현된 시스템은 SMP 방식으로 정확히 동작하였고 vCPU의 개수에 따라 성능향상을 보였다. 본 연구의 결과는 인공위성임무의 증가에 따른 위성비행소프트웨어의 복잡성을 해결하기 위한 가상화 시스템 도입에 기여할 수 있으며 고성능 병렬처리가 요구되는 가상화 환경에 적용되어 인공위성임무를 성공적으로 달성하는데 도움이 될 것이다.

References

1) Lim, S. S., "A Virtualization-based Software Architecture for Highly Secure and Reliable Mission Critical Systems," *Journal of Computing Science and Engineering,* Vol. 30, No. 3, 2012, pp. 47~54.

2) Ginosar, R., "Survey of Processors in Space," *Proceeding of Data* Systems in Aerospace, Dubrovnik, Croatia, May 2012, pp. 1~5.

3) Rufino, J. and Craveiro, J., "Robust Partitioning and Composability in ARINC 653 Conformant Real-Time Operating Systems," *1st*

INTERAC Research Network Plenary Workshop, Braga, Portugal, October 2008.

4) Carrascosa, E., Coronel, J., Masmano, M., Balbastre, P. and Crespo, A., "XtratuM Hypervisor Redesign for LEON4 Multicore Processor," *ACM SIGBED Review,* Vol. 11, No. 2, 2014, pp. 27~31.

5) Kaiser, R., "Combining Partitioning and Virtualization for Safety-Critical Systems," *SYSGO White Paper,* 2007.

6) Yoo, B. S., Choi, J. W., Jeong, J. Y. and Kim, S. W., "Performance Analysis of Processors for Next Generation Satellites," *IEMEK Journal of Embedded Systems and Applications,* Vol. 14, No. 1, February 2019, pp. 51~61.

7) *XtratuM in the Space Market*, Accessed on March 18, 2020, Online, Available: http://fentiss.com/rdi/missions.

8) Iqbal, S. M. Z., Liang, Y. and Grahn, H., "ParMiBench – An Open-Source Benchmark for Embedded Multiprocessor Systems," *in IEEE Computer Architecture Letters*, Vol. 9, No. 2, February 2010, pp. 45~48.

9) Guthaus, M. R., Ringenberg, J. S., Ernst, D., Austin, T. M., Mudge, T. and Brown, R. B., "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization,* Austin, TX, USA, 2001, pp. 3~14.

10) Liang, Y. and Iqbal, S. M. Z., "OpenMPBench – An Open-Source Benchmark for Multiprocessor Based Embedded Systems," *Master thesis report,* School of Computing, Blekinge Institute of Technology, Sweden, January 2010.