논문 2004-41CI-6-3

# 자바를 위한 분산된 병렬 컴퓨팅 환경

# (Distributed Parallel Computing Environment for Java)

이 상 윤*, 김 승 호**

( Sang-Yun Lee and Sung-Ho Kim)

요 약

자바의 쓰레드는 다중 처리 환경에서 하나의 프로그램 공간 내의 독립적인 프로세스로 취급되는 객체 요소이므로 병렬처리를 위한 독립적인 프로세스로 활용할 수 있다. 또한, 자바의 동기화 메커니즘과 쓰레드를 활용하면 병렬 처리를 수행하는 응용 프로그램을 쉽게 작성할 수 있다. 이에 따라, 자바의 병렬 처리 지원 기능을 분산된 컴퓨팅 환경에 적용하기 위한 많은 연구 결과가 있다. 본 논문에서는 레거시 자바 프로그램에 포함된 쓰레드를 분산된 컴퓨팅 환경에서 병렬 수행 하도록 지원하는 시스템 환경을 제안한다. TORB(Transparent Object Request Broker)라고 명명된 본 시스템은 프로그래밍 투명성을 지원하므로 이미 작성된 레거시 자바 프로그램을 간단한 변환 과정을 거친 후 병렬 수행 하도록 지원한다. TORB는 본 연구팀에서 이미 발표한 분산 프로그래밍 도구의 기능을 확장한 것이며, 이는 지정된 기능을 지정된 컴퓨터에서 수행하도록 지원하는 전형적인 분산처리 기능만을 보유하고 있었다.

Abstract

Since java thread is an object which is treated as independent process within one execution space in the multiprocessing environment, we can use it for independent process of parallel processing. Using thread and synchronization mechanism of java enables us to write parallel application program easily. Therefore, a lot of results are exist which is apply the feature of java that support parallel processing to the distributed computing environment. In this paper, we introduce a system of environment that support parallel execution of thread which is included in legacy java program. The system named TORB(Transparent Object Request Broker) enables us parallel execution of legacy java program after simple converting process, since it support the feature of programming transparency. TORB is extended version of distributed programming tool that is published by our research team. And it had only typical distributed processing feature that is execute a specified function at the specified computer.

**Keywords**: distributed parallel computing, programming transparency

## I. 서 론

자바의 쓰레드는 다중 처리 환경에서 하나의 프로그램 공간 내의 독립적인 프로세스로 취급되는 객체 요소이므로 병렬 처리를 위한 독립적인 프로세스로 활용할 수 있다. 또한, 자바는 쓰레드 상호간의 동기를 프로그래밍 언어 자체의 기능으로 제공하고 있으므로, 자바가 제공하는 동기화 메커니즘과 쓰레드는 병렬 처리를 수행하는 응용 프로그램을 작성하는데 상당한 역할을 담당할 수 있다.[1] 가상 기계에 기반을 둔 플랫폼 독립적인 실행 환경, 객체 지향 언어의 장점에 의한 프로그래밍 용이성, 범용성 있는 통신 관련 API의 지원 등은 병렬 처리 프로그래밍과 관련된 자바의 장점을 분산된 컴퓨팅 환경으로 확장할 수 있는 가능성을 뒷받침 해 준다. 이에 따라, 병렬 처리 응용 프로그램의 작성에 적합한 자바의 장점을 분산된 컴퓨팅 환경에 적용하기 위한 많은 연구 결과가 있다.[2,3,4,5,6] 이들은 자바로 작성된 응용 프로그램에 포함된 병렬 처리요소(쓰레드)를 분산된 컴퓨터에서 수행 할 수 있도록 독자적인 메커니즘을 제공하고 있으나 구현 방법, 동작 특성 및 지원 환경에 따

* 정회원, 대원과학대학 컴퓨터정보처리과
(Dept. of Computer Information Processing, Daewon Science College)
** 정회원, 경북대학교 컴퓨터공학과
(Dept. of Computer Engineering, Kyungpook National University)
접수일자: 2004년8월16일, 수정완료일: 2004년11월15일

른 제약 요소를 내포하고 있다.

본 논문에서는 동일한 구현 방안 중에서 상대적으로 적은 제약 요소를 가지며 프로그래밍 투명성을 지원하는 분산 병렬 컴퓨팅 환경을 제안한다. TORB(Transparent Object Request Broker)라고 명명된 본 시스템은 프로그래밍 투명성의 지원을 통하여 이미 작성된 레거시 자바 프로그램을 간단한 변환 과정을 거친 후 병렬 수행 하도록 지원한다. TORB는 본 연구팀에서 이미 발표한 분산 프로그래밍 도구의 기능을 확장한 것이며, 이는 지정된 기능을 지정된 컴퓨터에서 수행하도록 지원하는 전형 적인 분산처리 기능만을 보유하고 있었다.[7]

부가적인 지식 없이 작성한 프로그램의 병렬 처리 요소를 분산된 컴퓨팅 환경에서 수행 할 수 있도록 지원하는 실행 환경이 제공 된다면, 응용 프로그램 개발자는 작성 하고자 하는 프로그램의 자체 기능에 더욱 충실할 수 있다. 분산된 컴퓨팅 환경에 적용하기 위해 필요한 프로그래밍 지식은 개발 하고자 하는 응용 소프트웨어의 자체 기능과 상관없는 부담이며, 이것은 프로그래밍 투명성을 충분히 지원하는 분산 병렬 컴퓨팅 환경의 필요성을 뒷받침 해 준다. 프로그래밍 투명성이란 프로그래머가 특정 컴퓨팅 환경에 적용하기 위한 프로그램을 개발 할 때, 이미 익숙한 컴퓨팅 환경에서 동작하는 프로그램을 개발하는 것과 동일한 방법을 적용할 수 있도록 지원하는 것으로 설명될 수 있다.

TORB가 지원하는 응용 프로그램 개발의 투명성은 두 가지로 설명될 수 있으며, 하나는 응용 프로그램을 작성 할 때 부가적인 문법을 전혀 적용할 필요가 없다는 것이고, 다른 하나는 작성된 프로그램을 분산된 컴퓨팅 환경에서 실행할 때도 투명성이 제공된다는 것이다. 전자는, 응용 프로그램의 작성이 완료된 후 간단한 변환절차 만으로 분산된 컴퓨팅 환경에서 수행 되는 실행 코드로 변환하는 작업이 이루어지기 때문이고, 후자는 변환된 실행 코드를 분산된 컴퓨터로 배포하는 과정이 필요하지 않기 때문이다.

분산된 컴퓨팅 환경에서의 병렬 처리를 지원하는 TORB는 동적 메서드 호출에 의한 RMI (Remote Method Invocation) 메커니즘에 기반을 둔 방법으로 구현되어 있으며, 프로그래밍 투명성을 지원하는 분산된 병렬 컴퓨팅 환경을 제공하기 위하여 다음과 같은 몇 가지 특징을 가진다.

◆ 인수 전달 메커니즘의 투명성 확보를 위해서 JAVA Serialization과 JAVA Reflection을 사용한 동적 호출 기능을 사용하는 통합된 대행 객체를 정의하고 사용하여 원격 메서드 호출을 지원한다.

◆ 컴파일된 자바 클래스 파일로부터 분산 병렬 컴퓨팅 환경에 적합한 기능을 수행하는 수정된 자바 클래스 파일을 생성한다. 후 처리 만을 통하여 분산 병렬 컴퓨팅 환경에서 동작 하도록 변환하는 작업이 이루어지므로, 프로그램을 작성하는 동안 TORB의 사용법에 대한 지식이 없어도 된다.

◆ TORB가 제공하는 분산 병렬 컴퓨팅 환경은 P2P메커니즘에 기반을 둔 방법으로 구성되고 운영된다. 이는 참여하는 모든 컴퓨터에서 TORB에 포함되어 있는 서버를 기동하는 것만으로 TORB의 분산 병렬 컴퓨팅 환경이 형성되며, 이 환경에 참여하는 모든 컴퓨터에서 분산 병렬 컴퓨팅 환경을 활용하는 응용 소프트웨어를 실행할 수 있다는 것을 의미한다.

◆ 독자적인 자바 클래스 로더를 정의하여 활용 하므로 별도의 등록 절차 없이 분산 병렬 컴퓨팅 환경을 활용하는 응용 소프트웨어를 실행 할 수 있다. 이는 단일 컴퓨터에서 동작 하는 응용 프로그램을 후처리를 통하여 변환 한 후, 실행 하는 것만으로 여러 개의 컴퓨터로 분산 되어 수행됨을 의미한다. 즉, 다른 컴퓨터에 해당 프로그램의 실행 코드(클래스파일)를 미리 전송해 둘 필요가 없다.

본 논문은 프로그래밍 투명성의 제공에 의해 이미 작성된 응용 프로그램에 포함된 병렬 처리 요소를 분산된 컴퓨터를 통하여 병렬 수행하기 위한 분산 병렬 컴퓨팅 환경을 제안하기 위하여 다음과 같이 구성된다. 서론에 이어 Ⅱ장에서는 분산된 컴퓨팅 환경에서의 병렬 처리 지원과 관련 되는 내용 및 연구 결과를 소개하고 이들이 가지는 제약 사항과 범용성 및 프로그래밍 투명성의 문제점을 논의한 후 기능이 향상된 TORB의 특징을 제시한다. Ⅲ장에서는 분산된 컴퓨팅 환경에서 투명한 병렬 처리를 지원하기 위하여 제안된 TORB의 동작 메커니즘과 함께 구현 방안을 설명하고, Ⅳ장에서는 병렬 처리를 수행 하는 몇 가지 응용 프로그램을 TORB에 적용하여 실험한 결과를 분석 하며, Ⅴ장에서는 결론 및 향후 연구 계획을 제시 한다.

## Ⅱ. 자바와 관련된 분산 병렬 컴퓨팅 지원 환경

자바로 작성된 프로그램을 분산된 컴퓨팅 환경에서

| | |
|---|---|
| above JVM | Using Java's introspection feature<br>e.g., JavaParty |
| JVM | Implement JVM which has DSM feature<br>e.g. cJVM |
| below JVM | Implement JVM on existent DSM system<br>e.g., Java/DSM |

그림 1. 분산 병렬 컴퓨팅을 위한 세 가지 접근방안
Fig. 1. Three types of solution for distributed parallel computing.

병렬 수행 하는 환경을 제공하기 위한 접근 방안은 그림 1과 같이 설명될 수 있다.[3]

그림 1은 자바 가상 기계와 관련 하여 세 가지 단계로 나누어지는 접근 방안을 설명 하고 있으며, 이미 존재하는 DSM 시스템에서 동작 하는 자바 가상 기계를 구현 하는 방안과, DSM 기능을 제공 하는 변형된 자바 가상 기계를 구현 하는 방안, 자바의 Introspection 기능을 활용 하여 범용의 자바 가상 기계에서 동작 하는 분산 병렬 컴퓨팅 환경을 제공 하는 방안이 그것이다.

### 1. 분산된 컴퓨팅 환경에서의 자바 쓰레드

자바는 쓰레드 상호 간의 동기를 프로그래밍 언어 자체의 기능으로 제공 하고 있으므로, 자바가 제공 하는 동기화 메커니즘과 쓰레드는 병렬 처리를 수행 하는 응용 프로그램을 작성 하는데 상당한 역할을 담당 할 수 있다.[1] 그러나, 병렬 처리에 의한 수행 시간 단축의 효과는 자바 쓰레드가 여러 개의 프로세서에 의해 독립적으로 수행 된다는 가정 하에서 가능한 일이며, 이것은 여러 개의 프로세서가 장착된 컴퓨터에서 자바의 네이티브 쓰레드를 지원 하는 환경으로 제한된다. 분산된 컴퓨팅 환경을 구성 하는 대부분의 컴퓨터 들은 이기종의 컴퓨터로 구성 되며 한 개의 프로세서에 의해 처리되는 멀티프로세싱 환경이다. DSM환경에서 동작 하는 자바 가상 기계를 구현한 방법과 DSM기능이 구현된 변형된 자바 가상 기계에 의한 분산 병렬 컴퓨팅 환경은 병렬 처리를 위한 자바의 프로그래밍 범위를 제한하지는 않으나 플랫폼 독립적인 표준 가상 기계에 의한 자바의 실행 환경에 대한 범용성을 제한한다. 자바 실행 환경의 범용성은 분산된 컴퓨팅 환경에 참여 하는 컴퓨터 기종의 범위를 폭 넓게 확보 하는 효과를 제공 하므로 분산 병렬 컴퓨팅 환경을 구성 하는데 있어서 포기하기 어려운 장점이다. TORB는 자바의 표준 실행 환경에서 동작 하도록 구성 되었으므로 실행 환경에 대한 범용성을 유지 할 수 있으며, 통합된 대행 객체를 사용 하는 단일 객체 변환 기법과 자바 어셈블리 수준의 변환 방법을 적용하여 프로그래밍 투명성을 확보 하였다.

### 2. 관련연구

분산 병렬 컴퓨팅 환경을 제공하는 세 가지 접근 방안은 자바로 작성된 프로그램에 포함된 병렬 처리 요소를 분산된 컴퓨터에서 수행 하도록 지원 하는 동일한 목적을 가지지만 각각의 구현 방안 및 동작 특성에 따른 장단점을 내포하고 있다. 다음은 각각의 접근 방안에 따른 동작 특성 및 장단점이다.

- ◆ Java/DSM : Java/DSM은 기존의 TreadMarks /DSM 시스템에서 동작 하는 JDK-1.0.2 기반의 독자적인 자바 가상 기계를 구현 한 것으로서, 분산된 컴퓨팅 환경을 기존의 DSM시스템이 제공 하므로 쓰레드 및 동기화와 관련된 자바의 모든 기능을 투명하게 활용 할 수 있다는 장점이 있는 반면, 여기서 적용한 DSM 시스템을 적용할 수 있는 특정 컴퓨팅 환경에서만 동작 하므로 범용성 있는 활용을 위해서는 TreadMarks/DSM 시스템이 범용성을 가져야 한다.[2]
- ◆ cJVM : cJVM은 IBM IntelliStations로 구성된 Win/NT의 클러스터 컴퓨터에서 동작하며 분산 병렬 컴퓨팅 기능을 추가한 독자 적인 자바 가상 기계를 구현한 것으로서, Java/DSM에 버금가는 투명성을 지원 하는 장점이 있는 반면, 범용성 있는 활용을 위해서는 매번 새로운 운영체제에서 동작하는 동일한 기능의 자바 가상 기계를 구현 하여야 한다.[3]
- ◆ JavaParty : JavaParty는 범용의 자바 가상 기계에서 동작 하도록 구성된 서드파티 자바 패키지와 분산 병렬 컴퓨팅 환경의 지원을 위한 보조 프로그램들로 구성된다. 표준 자바 가상기계에서 동작 하도록 구성 하였으므로 실행 환경의 범용성은 확보 하였으나, 분산 시스템에 대한 변화된 환경을 완전하게 숨기지 못하였고, 원격 객체 상호간의 상속성만 허용하는 등의 제약성을 지니고 있다.[4]

### 3. TORB(Transparent Object Request Broker)

분산된 컴퓨팅 환경을 활용 하는 것은 네트워크로 연결된 여러 대의 컴퓨터가 상호 협조하며 동작해야 하는 본질 적인 응용문제를 다루는 것과 하나의 응용 프로그램이 분산되어 있는 컴퓨팅 자원을 통합하여 활용 하는

방안을 얻기 위한 것이라고 평가 할 수 있다. 인터넷 뱅킹을 비롯하여 온라인 게임, 인스턴트 메시징, P2P, 화상채팅 등 대표적인 오늘날의 응용 소프트웨어들은 네트워크로 연결된 여러 대의 컴퓨터가 상호 협조하여 동작해야하는 분산 처리의 본질적인 응용의 예로 생각할 수 있고, 단일 프로그램에 포함된 병렬 처리 요소가 네트워크로 연결된 여러 대의 컴퓨터를 활용 하는 방안을 제공하여 수행 시간 감소 효과를 얻는 것은 분산된 컴퓨팅 자원을 통합 하는 예로 생각할 수 있다.

### 가. TORB의 기능 향상

분산된 컴퓨팅 환경에서 여러 대의 컴퓨터가 협조 하며 동작해야 하는 소프트웨어의 개발을 지원하기 위하여 다양한 형태의 미들웨어들이 제안되어 있으며 성공적인 패러다임으로 객체 지향 분산 처리 기술이 주목받고 있다.[8] 그러나 기존의 분산 처리 지원 도구를 사용하는 프로그래밍은 대부분 각각의 솔루션에서 정의한 프로그래밍 패러다임을 학습 하고 적용해야 한다. 이러한 프로그래밍 패러다임은 새로운 형태의 문제 해결 방안을 창조 하는 경우도 있지만 단일 컴퓨팅 환경과 분산된 컴퓨팅 환경의 차이를 극복하기 위해 도입된 것이 상당한 비중을 차지하고 있으며 이를 학습 하여 개발 과정에 적용 하는 것은 작성 하고자 하는 프로그램의 자체 기능과 상관없는 부담이다.[8] 본 연구팀 에서는 프로그래밍 투명성을 지원하여 이러한 부담을 경감 시키고 상대적으로 높은 생산성을 얻을 수 있는 분산 처리 지원 도구를 발표한 바가 있으나 네트워크로 연결된 여러 대의 컴퓨터가 상호 협조하며 동작해야 하는 본질적인 응용을 지원 하는 것으로 제한되었다.[7]

단일 프로그램이 네트워크로 연결된 여러 대의 컴퓨터를 활용 하며 동작 하도록 지원 하는 것은 분산 되어 있는 컴퓨팅 자원을 통합 하여 활용 하는 것으로서 분산된 컴퓨팅 환경의 다른 활용이다. 이는 두 가지로 제공될 수 있으며 하나는 기존의 분산 처리 지원 도구를 활용 하여 프로그래머가 분산된 컴퓨팅 자원을 명시적으로 통합 할 수 있도록 지원 하는 것이고, 다른 하나는 분산된 컴퓨팅 환경을 전혀 고려하지 않고 작성된 프로그램이 분산 되어 있는 컴퓨팅 자원을 활용 하며 동작하도록 관련된 지원 방안을 마련하는 것이다. 전자는 분산된 컴퓨팅 자원을 프로그래머가 명시적으로 관리해야만 하므로 활용상의 부담을 내포 하고 있지만 기존의 분산처리 지원 도구가 적절한 것이면 충분히 지원할 수 있다. 이에 반해 후자는 이미 작성된 프로그램이 분산된 컴퓨팅 환경을 활용 하도록 지원 하는 것이므로 사용상의 용이함을 보장 받을 수 있으나 관련된 기능의 지원을 위한 부가 적인 조치가 필요하다.

기능이 향상된 TORB는 프로그래밍 투명성을 제공하는 기존의 분산 처리 지원 기능과 더불어 이미 작성된 자바 프로그램이 분산 되어 있는 컴퓨팅 자원을 활용하며 동작 하고 이를 통합 하는 기능을 투명하게 지원 하도록 재구성 하였다. 이는 Java/DSM, cJVM, Java-Party와 같은 기존의 지원 도구의 단점을 보완하기 위한 것이며, 기능 향상을 위한 재구성의 내용은 통합된 대행 객체의 도입, P2P 프로토콜의 적용을 통한 병렬 처리 환경의 구성, 독자적인 클래스로더의 적용을 통한 실행 방법의 투명성 제공 등이다.

### 나. TORB의 특징

TORB는 서드파티 자바 패키지와 분산 병렬 컴퓨팅 환경의 운영을 위한 보조 프로그램들로 구성되며 범용의 가상 기계에서 동작 하므로 자바의 범용성을 분산된 컴퓨팅 환경에서도 그대로 유지할 수 있다. 또한, 독특한 객체 변환 절차를 적용 하여 프로그래밍 투명성의 범위를 확장 하였고, 통합된 대행 객체를 통한 단일 객체 변환 기법을 적용 하여 실행 코드의 배포를 자동으로 처리할 수 있도록 하였다. 이에 대한 효과로 개발자는 프로그램을 작성하는 동안 TORB를 활용하기 위한 부가적인 문법을 전혀 사용 할 필요가 없고, 작성된 프로그램은 객체 변환 절차를 거쳐서 TORB가 지원 하는 보조 프로그램과 연동 하여 분산된 컴퓨팅 환경에서 병렬 처리를 수행한다. 이때, 변환된 실행 코드는 TORB가 제공 하는 독자적인 클래스로더에 의해 자동으로 배포 되어 동작 한다.

## III. 분산 컴퓨팅 환경에서의 병렬처리

TORB가 지원하는 프로그래밍 투명성이 병렬 처리 프로그램과 관련된 자바의 모든 기능을 수용 한다고 할 수는 없으나 동일한 접근 방법으로 구성된 기존의 분산 병렬 컴퓨팅 환경 보다 넓은 범위의 프로그래밍 투명성을 지원한다. III장에서는 실행 환경에 대한 범용성을 유지 하면서 분산된 컴퓨팅 환경에서의 투명한 병렬 처리를 제공하기 위하여 적용된 기법을 제시하여 TORB의 동작 메커니즘과 TORB가 제공 하는 프로그래밍 투명성의 범위를 논의 한다.

### 1. TORB 서버

자바는 객체 지향 프로그래밍 언어이므로 임의의 객체를 생성 하고 접근(access)하는 방법으로 프로그램이 실행된다. 분산된 컴퓨팅 환경에서 생성된 객체가 다른 컴퓨터에 존재 하는 것 이라면 이 객체에 대한 접근 조작은 네트워크를 통한 요청(request)과 응답(response)으로 처리 되어야 한다. TORB에서는 네트워크를 통하여 전달되는 이러한 요청을 처리 하고 응답하기 위해서 "TORB 서버"라는 서비스 프로그램을 도입 하였다. "TORB 서버"는 자바의 표준 실행 환경에서 동작 하므로 TORB가 제공 하는 분산된 컴퓨팅 환경은 자바의 실행 환경에 대한 범용성을 유지 할 수 있다. "TORB 서버"는 네트워크를 통하여 전달되는 특정 객체에 대한 요청을 독립 적인 쓰레드에 의해 처리하여 여러 개의 객체에 대한 접근 조작을 동시에 처리 할 수 있도록 하여야 한다.

"TORB 서버"는 원격 컴퓨터에서 생성된 객체에 대한 접근 조작을 지원 하는 기능을 수행 하므로 작성된 프로그램이 실행을 시작하기 전에 동작 중 이어야 한다. 따라서 분산 처리에 참여 하는 모든 호스트에 "TORB 서버"를 기동 하는 조치가 필요 하다. TORB에서는 "TORB 서버"를 기동하기 위하여 명시적인 방법과 묵시적인 방법을 제공 한다. 명시적인 방법은 사용자가 명령 라인에서 직접 서버를 기동 하는 방법 이고, 묵시적인 방법은 분산된 환경 에서 동작 하도록 변환 할 때 "TORB 서버"를 중복되지 않게 기동 하는 코드를 강제로 삽입 하여, 개발자가 작성한 프로그램이 실행을 시작 할 때 자동으로 기동 하도록 하는 방법이다.

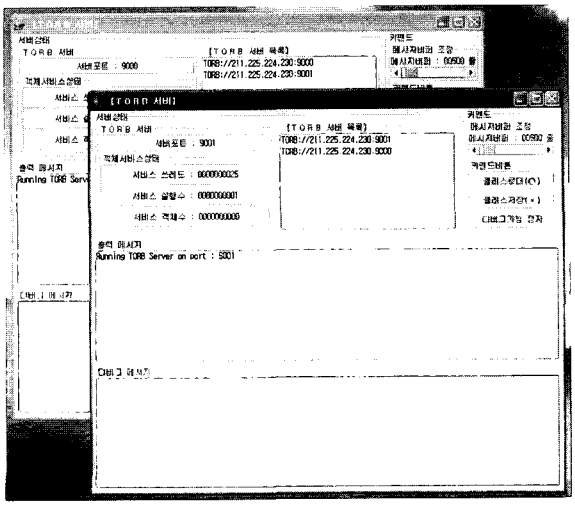

그림 2. "TORB 서버" 기동화면
Fig. 2. Screen shot of "TORB Server".

명시적인 방법으로 기동하는 "TORB 서버"는 분산 병렬 컴퓨팅 환경을 구성하기 위하여 각각의 분산된 컴퓨터에서 사용자가 기동 하는 것이다. 그림2는 명시적인 방법으로 2개의 "TORB 서버"를 기동한 화면 이다. 그림2에 나타난 바와 같이 기동한 "TORB 서버"들은 P2P 기반의 메시지 교환에 의해 "TORB 서버" 상호간의 URL을 공유 하고 있고, 생성된 원격 객체의 수, 서비스 쓰레드의 상태, 원격 메서드 호출 횟수 등 다양한 정보를 표현 하고 있다.

### 2. 분산 병렬 컴퓨팅 환경의 형성

병렬 처리를 위하여 자바로 작성한 프로그램이 병렬 동작을 위하여 독립 적인 쓰레드를 생성 할 때, 이들 쓰레드는 분산된 컴퓨팅 환경을 구성 하는 서로 다른 컴퓨터에서 생성 및 접근 되도록 구성 하여야 병렬 처리에 의한 수행 시간 감소 효과를 기대 할 수 있다. 개발자가 작성한 프로그램이 수행 되는 동안 병렬 처리 요소를 분산된 컴퓨터를 통하여 수행 하도록 하기 위하여 가용한 원격 컴퓨터들의 목록이 필요하며 이는 다음의 과정을 통하여 유지 및 사용된다.

병렬 처리를 지원하기 위하여 분산된 컴퓨팅 환경을 구성 하는 과정은 그림 3과 그림 4에 명시된 형태로 설명할 수 있다. 그림3은 "TORB 서버"를 기동 하거나 수행 종료 할 때, 이미 동작 중인 다른 "TORB 서버"와의 메시지 교환을 통하여 분산 병렬 컴퓨팅 환경을 구성하기 위하여 참여 하는 "TORB 서버"의 목록을 갱신 하는 과정을 나타낸 것이다. 새로운 "TORB 서버"를 기동할 때, 이미 동작 중인 "TORB 서버"중 하나를 선택해서 "ping-1"메시지를 전송 하고, "ping-1"메시지를 전달 받은 "TORB 서버"는 모든 나머지 "TORB 서버"에게 "pong"메시지를 전송 하여 각자가 분산 병렬 컴퓨팅 환경에 참여 하는 "TORB 서버"의 목록을 갱신 하도록 한다. 이는 분산 병렬 컴퓨팅 환경을 구성 하기 위해

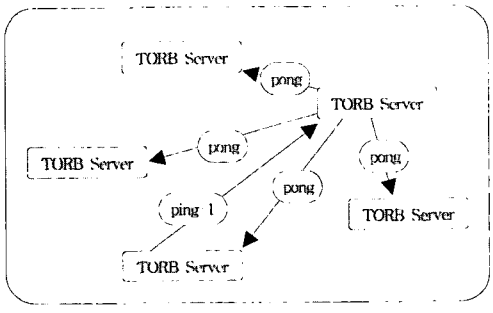


그림 3. 분산 병렬 컴퓨팅 환경의 형성
Fig. 3. Conformation for parallel DCE.

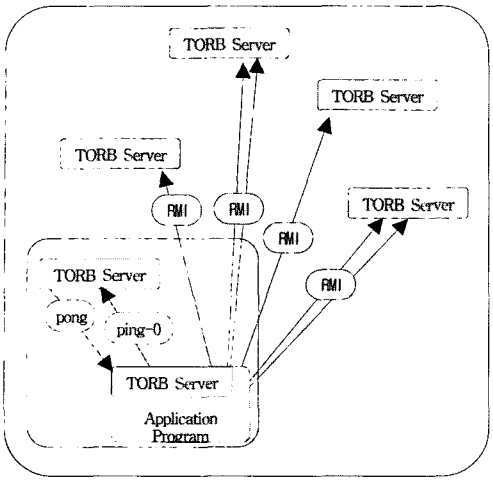


그림 4. 원격 메서드 호출에 의한 병렬 처리
Fig. 4. Parallel processing by RMI.

GNU-Tella에 정의된 프로토콜의 일부 기능을 차용 한 것으로서 TORB가 제공 하는 분산된 병렬 컴퓨팅 환경이 P2P에 기반을 둔 형태로 구성 및 운영 될 수 있음을 의미 한다.[9]

그림4는 이미 형성된 하나의 분산 병렬 컴퓨팅 환경에서 병렬 처리를 수행 하는 응용 프로그램이 동작 하는 과정을 표현 하고 있다. 분산 병렬 컴퓨팅 환경에 참여 하는 컴퓨터에는 하나 이상의 "TORB 서버"가 동작 중이고, 후 처리를 통하여 변환된 응용 프로그램은 이 "TORB 서버"에게 "ping-0"메시지를 보내어 동작 중인 "TORB 서버"의 목록을 "pong"메시지를 통하여 받는다. 이는 후 처리 과정 에서 응용 프로그램 자체에 묵시적인 방법으로 "TORB 서버"를 기동 하는 코드를 삽입 하도록 하여 구현 할 수 있다. 후 처리를 통하여 변환된 응용 프로그램은 확보한 "TORB 서버"목록을 이용 하여 병렬 처리에 필요한 만큼의 객체를 원격의 컴퓨터에 생성 한 후, 이들에 대한 접근(원격 메서드 호출)을 통하여 수행 한다. 분산 병렬 컴퓨팅 환경에 참여 하는 모든 컴퓨터는 동등한 기능을 발휘 할 수 있으므로, 다른 컴퓨터 에서도 동일한 방법으로 병렬 처리를 수행 하는 응용 프로그램을 실행 할 수 있다.

### 3. 대행객체

병렬 처리를 수행 할 목적으로 작성된 응용 프로그램이 분산된 컴퓨팅 환경을 활용하기 위해서는 적절한 변환이 선행 되어야 한다. TORB에서는 자바 어셈블리 수준의 변환 기법을 적용 하여 이미 작성된 자바 프로그램을 분산된 컴퓨팅 환경을 활용 하는 버전으로 변환 하는 후 처리 프로그램을 제공 한다. 이때 체계적인 변환 기법과 분산된 컴퓨팅 환경 에서의 자바 객체에 대한 투명성을 지원 하기 위해서 통합된 대행 객체를 도입 하였다. 통합된 대행 객체는 변환 과정에서 임의의 자바 객체에 대한 개별 적인 대행 객체를 생성 하고 이를 통한 분산 처리를 수행 하는 방법과는 구별 되므로 후 처리 과정 에서 대행 객체를 생성 하는 절차가 필요 없다.[4,7,10,11] Ⅳ장 3절 에서는 TORB를 구현하기 위하여 도입한 통합된 대행 객체가 갖추어야 하는 기능과 구현 방안 및 부수 적인 효과를 논의 한다.

#### 가. 대행 객체의 기능 및 구현 방안

대행 객체는 분산된 컴퓨팅 환경과 관련 없이 작성된 임의의 자바 객체에 대한 모든 접근 조작을 대신 수행 하여 원격 컴퓨터에 존재 하는 해당 객체에 대한 서비스를 처리 하는 기능을 갖추어야 한다. 이를 위하여 임의의 자바 객체에 대하여 발생할 수 있는 모든 접근 조작을 규정 하고 이들을 대신 하여 처리 할 수 있는 통합된 대행 객체를 도입하였다. 다음은 본 논문에서 규정한 단일 컴퓨팅 환경 에서의 임의의 자바 객체에 대한 접근 조작들과 이들을 처리하기 위한 대행 객체의 구현 방안 이다.

◆ Object creation : 자바는 키워드 "new"를 사용하여 임의의 객체의 클래스에 정의되어 있는 생성자를 호출 하여 인스턴스를 생성 하며 그 객체의 참조 정보를 저장 한다. 저장된 참조 정보는 향후 이 객체에 대한 접근 조작이 필요 할 때 마다 사용 된다. 대행 객체는 네트워크를 통하여 객체 생성에 대한 요청을 보내어 필요한 객체가 원격 컴퓨터에서 생성 되도록 한 후 이 객체에 대한 모든 접근 조작을 대신 하여 수행 할 수 있어야 한다. 이를 위하여 Java reflection 기법에 의한 동적 호출 방법을 적용 하였으며 원격 객체에 대한 메서드를 호출 하는 경우에도 동일한 방법이 사용 된다.

◆ Method invocation : 자바에서는 일반적으로 연산자 "."을 사용 하는 정적인 형태의 메서드 호출 방법이 사용 된다. TORB에서는 원격 에서 이미 실행 중인 "TORB 서버"에 의해 해당 객체의 메서드를 실행해야 하므로 동적 호출 방법이 적용 되어야 하며 이는 객체 생성의 경우와 동일하다. 따라서 정적 호출 방법으로 작성된 일반 적인 자바 실행 코드를 동적 호출 방법을 적용한 코드로 변환 하는 조치가 필요 하다. 이는 자바 어셈블리 수준의 변환 기법에 따라

자바의 실행 시간 스택을 적절히 조정 하고, Java serialization 기법에 의한 객체 전송 방법을 적용한 인수 전달 방법을 사용 하여 구현 하였다.

◆ Treat as parameter : 임의의 객체에 대하여 자바가 지원 하는 인수 전달 메커니즘은 객체의 참조 값을 전달하는 방법을 적용 하고 있다. 분산되어 있는 다른 컴퓨터에 직접적인 참조 값을 전달하는 것은 불가능 하므로 대행 객체를 전송 하는 방법을 적용하였다. 따라서 대행 객체는 임의의 원격객체에 대한 참조 정보의 역할을 담당 하여야 하며 네트워크를 통하여 다른 컴퓨터로 전송될 수 있도록 구현하였다. 네트워크로 연결된 다른 컴퓨터에 임의의 자바 객체를 전송하기 위해서는 Java serialization에 적용 할 수 있어야 한다.

◆ Treat as returning reference : 자바에서는 임의의 객체가 메서드를 실행 한 결과 값으로 취급 되는 경우에 해당 객체의 참조 값을 전달한다. 대행 객체는 임의의 원격 객체에 대한 완전한 참조 정보의 역할을 담당 하도록 구현 되는 것이므로 인수 전달 메커니즘의 해결 방법과 동일한 기법을 적용하여 해결 하였다.

◆ Java synchronization : 자바에서는 쓰레드간의 동기화와 관련된 기능을 언어 자체의 기능으로 제공하고 있으며 동기화와 관련 되는 객체를 운영체제에 의해 할당된 하나의 프로세스만이 접근 하도록 제한 또는 해제 하는 방법으로 지원 한다. 다른 컴퓨터에 존재하는 객체와의 동기화를 투명하게 제공하는 방안은 KaRMI에 소개되어 있으며, TORB에서는 원격 객체 관리자를 도입 하고 쓰레드의 고유한 식별 정보를 하나의 원격 메서드 호출이 완료 될 때 까지 유지하는 등의 독자 적인 방법으로 해결 하였다.[4,12]

◆ Networking feature : 대행 객체는 네트워크를 통한 메시지 송수신에 의하여 다른 컴퓨터에 존재하는 자바 객체에 대한 접근 조작을 처리 하므로 통신기능이 필연적으로 포함 되어야 한다. 일반적으로 네트워크 기능을 지원 하는 자바의 클래스 라이브러리는 기계 종속적인 방법으로 구현 되어 있으므로 Java serialization에 의해 다른 컴퓨터로 전송되어야 하는 대행 객체의 멤버 데이터에 포함 될 수 없다. 이 사실은 대행 객체에 포함 될 수 있는 멤버 데이터를 제한 하지만 독특한 코딩 기술을 적용 하여 원격 객체와의 네트워크를 통한 연결 정보를 유지하며 다른 컴퓨터로 전송 될 수 있는 대행 객체를 구현 하였다.

◆ Dynamically specified distributed processing feature : 대행 객체의 메서드 정의에 URL을 지정하는 예약된 메서드를 추가해 두면 원격 객체가 생성 되고 실행될 호스트를 실행 시간에 동적으로 지정할 수 있다. 프로그래머는 프로그램 코드 내에서 이 메서드의 원형(prototype)만 정의한 후, 이 메서드를 호출 하면 되기 때문이다. 즉, 임의의 객체에 대한 URL을 설정하기 위하여 예약된 메서드를 호출 하는 실행 코드는 개발자가 그 객체에 마련 해 둔 메서드를 호출 하는 것이 아니라 대행 객체에 이미 마련되어 있는 메서드를 호출 하는 코드로 변환 된다. 이 기능은 특정 기능을 수행할 컴퓨터를 실행 시간에 동적으로 명시 하는 기능이 필요할 때 에만 적용 되는 것이므로 프로그래밍 투명성을 해친다고 볼 수 없다.

### 나. 통합된 대행객체와 클래스로더

#### (1) 클래스로더

분산된 컴퓨팅 환경 에서 수행 할 목적으로 작성 된 응용 프로그램이라 할지라도 그의 실행 코드(클래스파일)는 프로그램 개발자의 컴퓨터 에만 존재한다. CORBA의 규약을 따르는 일반 적인 분산 처리 환경에서는 응용 프로그램을 수행하기 전에 구현 객체(실행코드)에 대한 IDL을 정의 한 후 구현 저장소(Implementation Repository)에 등록 하여야 하고, 네이밍서비스 및 트레이더서비스 등 과 관련 되어 운영 된다.[13] 이러한 절차는 서로 다른 개발 언어로 구성 된 컴퓨팅 환경을 통합 하는 분산 처리 환경을 구성하기 위한 방안으로 적절 한 것으로 평가 할 수 있으나 단일 개발 언어로 구성된 컴퓨팅 환경을 적용 하는 경우에는 사용자에게 부수적인 오버헤드 이다. TORB는 단일 언어를 위한 분산 병렬 컴퓨팅 환경이므로 이러한 절차를 없애고 더욱 투명한 실행 환경을 제공한다. 이러한 투명한 실행 환경을 제공하기 위하여 TORB에서는 독자적인 클래스 로더를 설계 하고 적용 하였다. 그 결과로, 사용자는 개발한 프로그램의 실행 코드를 원격 컴퓨터에 미리 전송 하거나 시스템 환경에 등록 하는 절차를 수행할 필요가 없다. 원격 컴퓨터에서 필요로 하는 프로그램의 실행 코드는 독자적인 클래스 로더에 의해 실행 시간에 동적인 방법으로 프로그램을 기동한 컴퓨터로부터 전송 된다.

(2) 통합된 대행객체

본 연구팀 에서는 이미 작성된 자바 프로그램을 분산된 컴퓨팅 환경을 활용 하는 버전으로 변환 하는 동안 원격 객체에 대한 개별적인 대행 객체를 생성 하는 분산처리 지원 도구를 발표 하였으며 기존의 많은 분산처리 지원 도구 들이 유사한 방법을 적용 하고 있다.[4,7,10,11] 통합된 대행 객체를 도입함에 따라 분산된 컴퓨팅 환경을 활용하는 버전으로 변환 하는 동안 대행 객체를 생성 할 필요가 없고, 이미 작성된 자바 프로그램의 실행 코드(클래스파일)에 대한 변환 조작만이 적용 된다. 자바 프로그램의 실행 코드를 분산된 컴퓨터에 동적인 방법으로 배포하기 위해서는 독자적인 클래스로더를 도입 하여야 하지만, 분산된 컴퓨팅 환경에서 동작 하도록 변환된 실행 코드가 다른 객체 클래스(개별적인 대행 객체)의 실행 코드와 연계되는 경우 에는 "NoClassDefFoundError"라는 예외 상황의 발생을 해결하기 어렵다. TORB에서는 통합된 대행 객체를 도입하여 이미 작성된 자바 프로그램의 실행 코드를 단일 객체 변환 기법을 적용하여 분산된 컴퓨팅 환경에서 동작하도록 변환 하므로 이러한 예외 상황을 피할 수 있다.

## 4. 실행코드(클래스파일)의 변환

### 가. 자바 어셈블리와 변환도구

프로그래밍 투명성의 지원에 의하면 분산된 컴퓨팅 환경 에서의 병렬 처리를 목적으로 작성 하는 프로그램이라 할지라도 개발자는 단일 컴퓨팅 환경만을 고려하면서 프로그램을 작성 할 수 있다. 이는 TORB가 이미 작성된 자바 프로그램을 전혀 수정 하지 않고 분산된 컴퓨팅 환경에서 수행 하도록 변환 하는 후 처리 도구를 제공하기 때문이다. 따라서 TORB가 제공 하는 분산 병렬 컴퓨팅 환경 에서 수행 할 목적으로 프로그램을 작성 하는 동안, 프로그래머는 자신이 병렬 처리 혹은 분산 처리 프로그램을 작성 하고 있다는 사실과 TORB가 제공 하는 프로그래밍 투명성의 범위에 대한 인식만이 추가로 필요 할 뿐이다.

"TORBC"라고 명명된 후 처리 도구는 자바를 위한 표준 실행 환경에서 동작 하며 단순한 형식의 설정 정보에 의해 변환 작업을 수행 한다. "TORBC"에 의해 수행 되는 변환 작업은 BCEL의 Jasmin과 JasminVis itor을 활용 하여 자바 어셈블리 수준 에서 이루어지도록 구현 하였으며, 변환 하고자 하는 자바 클래스 파일에 대한 자바 어셈블리 코드를 JasminVisitor에 의해 생성 하고, 의미 분석과 변환 과정을 통하여 수정된 자바

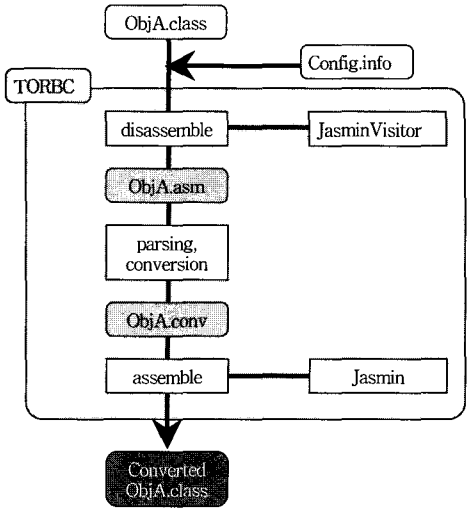


그림 5. TORBC에 의한 변환과정
Fig. 5. Converting process by TORBC.

```
Third=TORB://remoteA.ac.kr:9000
Server=TORB://remoteA.ac.kr:9000
Client=TORB://remoteB.ac.kr:9001
Go=self
TSP=any
```

그림 6. 설정 파일의 예
Fig. 6. Example of configuration file.

어셈블리 코드를 생성 한 후, Jasmin에 의해 변환된 자바 클래스 파일을 생성 하는 절차로 구성 된다.[14] 그림 5는 TORBC가 이미 작성된 자바 프로그램을 분산된 컴퓨팅 환경을 활용 하는 버전으로 변환 하는 과정을 설명 하고 있다.

### 나. 설정파일

TORB의 후처리 도구는 그림9와 같은 설정 파일을 후 처리를 위한 정보로 사용 한다. 이는 이미 작성된 프로그램 중 분산 혹은 병렬 처리의 역할을 담당 하는 자바 객체 들이 생성될 호스트를 명시 하고 이들과 관련되어 동작 하는 다른 자바 객체들의 목록을 명시하기 위한 것이다.

설정 파일의 내용은 그림 6와 같이 원격 객체의 이름과 URL의 순서쌍으로 구성 되며, 자바의 java.util.Properties에 적용 하는 형식 이다. 여기에서 URL을 "self"로 지정 하면 로컬 호스트에서 생성 되는 객체 이며 원격 객체를 접근하는 코드가 포함되어 있다는 의미이고, "any"로 지정 하면 "TORB 서버"의 서버 할당 알고리즘에 의해 로컬 컴퓨터에 기동된 "TORB 서버"에 등록된 모든 "TORB 서버"중 하나를 선택 하여 생성 되고 실행되어야 한다는 의미 이며, 이로 인해 불특정 다수의 컴퓨터를 통한 병렬 처리가 가능해 진다. 특히, URL

이 특정한 주소로 지정되는 경우는 특정 기능을 지정된 컴퓨터에서 실행 하도록 명시 하는 분산 처리 기능을 목적으로 사용 할 때 적용 한다.

### 다. 의미 분석 및 변환

"TORBC"가 수행 하는 의미 분석 및 변환 작업은 자바 어셈블리 수준에서 이루어지며 원격 컴퓨터 에서 생성 되는 객체에 대한 접근을 수행 하는 어셈블리 코드를 통합된 대행 객체를 통하여 동일한 효과를 발휘 할 수 있도록 하는 것이다. 이를 위하여 "TORBC"가 처리 하는 변환 작업은 다음과 같다.

#### (1) 객체생성 및 메서드 호출

그림7에는 임의의 객체를 생성 하고 메서드를 호출 하는 간단한 자바 프로그램을 제시 하였고, 반전된 부분의 코드는 그림8의 자바 어셈블리 코드로 표현 된다. 이를 통하여 표준 자바 실행 환경 에서 임의의 자바 객체를 생성 하는 과정을 알 수 있고, 변환 절차를 규정 할 수 있다. 그림8에 따르면, 자바 어셈블리 코드에 의한 객체의 생성은 다음과 같이 진행 되며, 이 객체가 다른 컴퓨터 에서 생성 되어 동일한 효과를 발휘 할 수 있도록 각각에 대한 변환 조작이 적용 된다.

◆ Create object reference : 니모닉 코드 "new"에 의하여 객체의 참조 값을 스택에 저장한 다. 분산된 컴퓨팅 환경 에서 이 객체는 다른 컴퓨터 에서 생성 되어야 하므로 통합된 대행 객체에 대한 참조 값을 스택에 저장 하도록 대치한다. 이는 단순히 문자열을 대치하는 조작으로 충분 하다.

◆ Load parameters onto stack : 생성자를 호출하기 위한 인수 값을 스택에 저장한다. 이는 해당 객체에 대한 생성자를 호출 할 때 적용 하는 인수들을 전달 하기 위한 것이며 이 단계에 대한 변환 조작은 필요 하지 않다.

◆ Invoke constructor : 니모닉 코드 "invokespecial"에 의하여 생성자를 호출 한다. 이 니모닉 코드는 해당 객체의 생성자를 호출하기 위한 것으로서 스택에 저장된 인수 들을 사용 하는 정적 호출 방법이 적용된다. 분산된 컴퓨팅 환경에서 이 객체에 대한 생성자는 다른 컴퓨터 에서 실행되어야 하므로 스택에 저장된 모든 인수 들은 네트워크를 통하여 전송 되어야 한다. 이를 위하여 스택에 저장된 인수 들을 추출 하여 배열 변수에 저장 하는 절차가 필요 하며 TORB에서는 이러한 변환 조작을 수행하는 자바 어셈블리 코드를 정의 하여 적용 하였다. 이들은 자바의 실행 시간 스택을 직접 조작 하는 일련의 어셈블리 코드로 구성 된다. 이에 따라 스택에 저장된 원시 인수 들을 배열 변수로 옮겨서 통합된 대행 객체의 생성자를 호출하기 위한 인수로 활용 한다. 대행 객체의 생성자는 배열에 저장된 원시 인수 들을 네트워크를 통하여 원격 컴퓨터에 전송 하고, 원격 컴퓨터 에서는 전송 받은 원시 인수를 사용하여 해당 객체의 생성자를 동적으로 실행 한다. 이러한 일련의 절차는 로컬 컴퓨터에서의 정적 호출에 대한 어셈블리 코드를 원격 컴퓨터에서의 동적 호출로 변환 하는 과정이며 그림9와 같은 절차로 정리 된다. 자바 어셈블리 코드를 분석해 보면 임의의 객체에 대한 생성자 호출과 일반적인 메서드 호출이 거의 유사한 절차를 수행 한다는 것을 알 수 있다. 따라서 원격 컴퓨터에 존재하는 객체에 대하여 일반적인 메서드를 호출하는 경우에도 동일한 변환 절차를 적용할 수 있다.

```
class ObjA {
    public static void main(String args[]) {
        ObjB oB = new ObjB("Batman");
        oB.greeting();
    }
}
```

```
class ObjB {
    String person;
    ObjB(String name) {
        this.person = name;
    }
    void greeting() {
        System.out.println("Hello, "+person);
    }
}
```

그림 7. 자바 소스 코드
Fig. 7. Java source code.

```
.method public static main([Ljava/lang/String;)V
.limit stack 3
.limit locals 2
.var 0 is arg0 [Ljava/lang/String; from Label0 to Label1

Label0:
.line 3
        new ObjB
        dup
        ldc "Batman"
        invokespecial ObjB/<init>(Ljava/lang/String;)V
        astore_1
.line 4
        aload_1
        invokevirtual ObjB/greeting()V
Label1:
.line 5
        return

.end method
```

그림 8. 자바 어셈블리 코드
Fig. 8. Java assembly code.

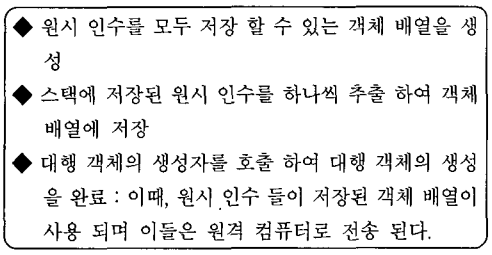


그림 9. 동적 호출을 위한 변환
Fig. 9. Converting for dynamic invocation.

(2) 키워드"this"에 대한 투명성

자바를 포함한 일반적인 객체 지향 프로그래밍 언어에서 사용되는 키워드 "this"는 객체 인스턴스 자신을 지칭 하는 용도로 광범위 하게 사용 된다. 자바 어셈블리 코드를 분석해 보면 모든 동적 메서드 내에 객체 자신을 참조 하는 로컬 변수가 미리 할당되어 있으며 키워드 "this"가 사용 될 때 마다 이 로컬 변수를 활용 하는 것을 알 수 있다. 분산된 컴퓨팅 환경에서 키워드 "this"에 의해서 임의의 객체가 다른 컴퓨터로 전송되어야 하는 경우 에는 대행 객체를 전송 하는 것이 유일한 방법 이므로 객체 자신에 대한 대행자 역할을 수행하는 대행 객체를 멤버 변수로 추가로 삽입 하고 이를 전송 하는 방법으로 원격 컴퓨터에서의 이 객체에 대한 접근을 처리 한다. 원격 컴퓨터 에서는 전송 받은 대행 객체를 통하여 이 객체를 접근해야 하므로 동일한 실행 공간에서 동작 중인 "TORB 서버"가 필요 하다. 이는 분산 되어 처리 되어야 하는 객체의 모든 생성자에 "TORB 서버"를 중복 되지 않게 기동 하는 어셈블리 코드를 삽입 하여 해결 할 수 있다. 이로 인하여, 로컬 컴퓨터에 생성된 객체가 원격의 컴퓨터에서 접근될 때 통합된 대행 객체를 통하여 처리 될 수 있으며 "TORBC"는 이와 관련된 적절한 변환 처리를 수행 한다.

(3) 인수전달의 투명성

자바는 단위 데이터(primitive data)의 경우 에는 복사에 의한 방법, 복합 데이터(객체, 배열 등)의 경우에는 참조에 의한 방법의 인수 전달 메커니즘을 적용 한다. 분산된 컴퓨팅 환경에서 단위 데이터에 대한 인수 전달은 Java serialization 기능을 활용 하여 쉽게 구현 할 수 있으나 복합 데이터에 대한 인수 전달은 참조에 의한 방법을 적용 하는 인수 전달 메커니즘이 적용 되어야 하므로 특별한 기법을 도입해야 한다. TORB에 적용된 통합e된 대행 객체는 임의의 객체를 네트워크를 통하여 참조 하는 역할을 수행 하도록 설계 되었으므로 대행 객체를 전송 하여 객체에 대한 참조 정보를 전달 하는 효과를 발휘 한다. 이에 따라, 자바의 메서드 호출 시 인수로 취급되는 임의의 객체에 대한 어셈블리 코드는 대행 객체를 전달하는 코드로 적절히 변환 한다. 또한, 특정 메서드를 수행 한 결과 값으로 취급 되는 객체에 대한 변환 처리도 동일한 방법 으로 처리 한다.

(4) 멤버 데이터의 접근

자바 프로그램에서 객체에 포함된 멤버 데이터는 연산자 "."를 사용 하여 접근 하는 방법과 해당 멤버 데이터를 접근 하는 메서드를 통하여 접근 하는 방법을 모두 사용 할 수 있다. 멤버 데이터를 접근 하는 메서드를 작성 하는 것은 번거로운 작업 이므로 일반 적인 경우에는 연산자 "."를 사용 한다. 그러나 멤버 데이터를 접근 하는 메서드는 구조가 단순 하므로 "TORBC"와 같은 자동화 도구에 의해 작성 될 수 있다. 이에 따라, "TORBC"는 임의의 객체에 포함된 모든 멤버 데이터를 접근 하는 메서드(_get_?, _put_?)에 대한 어셈블리 코드를 객체 정의에 추가 한다. 자바 어셈블리 코드를 분석해 보면, 임의의 객체에 포함된 멤버 데이터를 접근 하는 코드는 니모닉 코드 "getfield"와 "putfield"에 의해 처리 된다. "TORBC"는 자바 어셈블리 수준의 변환 처리를 수행 하므로 이들을 모두 메쏘들 호출에 의한 코드로 변환 할 수 있다. 이때 TORB에 적용된 동적 메서드 호출 기법과 인수 전달 기법을 활용 하여 원격 객체에 대한 멤버 데이터를 접근 하는 효과를 발휘 하도록 할 수 있으며 이미 작성된 자바 프로그램에 대한 투명성을 유지 한다.

(5) 원격접근과 로컬접근의 구별

"TORBC"에 의한 변환 조작은 원격 컴퓨터에 존재하는 객체를 접근 하는 어셈블리 코드를 대상으로 선택적으로 적용 되어야 한다. 이를 위하여, 자바 어셈블리 코드가 원격 객체를 접근 하는 코드인지 로컬의 객체를 접근 하는 코드 인지를 구별 하여야 한다. "TORBC"는 이 기능의 체계적인 구현을 위하여 어셈블리 코드 상의 스택 상태를 추적 하는 방법을 사용 한다. 이는 자바 어셈블리의 모든 니모닉(mnemonic)코드가 자바의 실행 시간 스택을 갱신 하는 과정을 추적 하여 가상의 자바 스택 정보를 생성 및 갱신 하며 이를 이용하는 방법 이다. 이는 자바에 적용된 모든 메서드가 실행 시간 동안 미리 할당된 스택의 범위 내에서만 독립적으로 동작하

기 때문에 적절한 방법 이다. 그리고 이러한 사실은 자바 어셈블리 코드에 대한 분석을 통하여 확인 할 수 있다.[15]

지금까지 논의된 변환 조작을 통하여 이미 작성된 자바 프로그램을 분산된 컴퓨팅 환경을 활용 하는 버전으로 자동 변환 할 수 있으며 충분한 수준의 프로그래밍 투명성을 제공 할 수 있다. 본 연구팀은 제시된 변환 조작에 의해서 처리될 수 없는 경우가 발견 된다 할지라도 많은 경우에 적절한 변환 방안을 찾을 수 있다고 확신 하고 있으며, 이는 자바 어셈블리 수준의 변환 메커니즘을 적용 하고 있기 때문 이다.

### 라. 제한사항

TORB에서는 분산된 컴퓨팅 환경에 존재 하는 객체에 대한 접근을 지원 하기 위하여 통합된 대행 객체를 도입 하였고, 단일 객체 변환 기법을 적용 하였다. 그 결과로 병렬 처리 응용 프로그램의 작성에 적합한 자바의 장점을 분산된 컴퓨팅 환경에 적용하기 위한 투명성을 상당히 확보 하였으나 컴퓨팅 환경의 차이로 인하여 수용 하지 못하는 경우도 존재할 수 있다. 현재, TORB에 의해 지원 되는 원격 객체는 다음과 같은 한계를 지니게 되며 향후 적절한 변환 방안을 적용할 예정이다.

◆ 다형성이 지원되지 않는 경우 : 객체 지향 프로그래밍 언어의 다형성에 의해 동일한 메서드에 대한 처리 동작이 객체의 인스턴스에 따라 다르게 수행되어야 하는 경우를 손쉽게 지원 할 수 있으며 TORB가 지원 하는 분산 컴퓨팅 환경 에서도 거의 대부분 적용할 수 있다. 그러나 원격 객체로 취급되는 임의의 객체가 JDK의 객체 라이브러리에 존재 하는 임의의 객체를 상속 받은 경우이고, 원격 객체 자신이 인수가 되면서, 다형성을 위해 부모 객체에 정의되어 있는 메서드를 호출 하는 경우에는 변환된 코드가 동작 할 수 없다. 원인은 부모 객체에 정의되어 있는 메서드에 전달하는 인수가 해당 원격 객체와 상위 상속 관계에 있는 객체이기 때문 이며 이들 상위 객체를 원격 객체로 취급 하지 않았기 때문이다. 설정 파일에 명시하기만 하면 이들 상위 객체에 대한 변환된 클래스 파일을 생성 하여 적절한 동작을 수행하는 라이브러리 객체로 변환 할 수는 있지만, JDK의 라이브러리에 포함 되어 있는 변환 되기 전의 클래스 파일과 대치 하여야 하는데 이는 현실적으로 불가능 하다. 이 경우 새로운 형태의 변환 기법이 적용 되어야 하는데 원인이 되는 메서드의 원형(prototype)을 적절히 변경 하여 원격 객체에 강제로 삽입 하는 방법 외에 몇 가지 변환 기법을 적용할 예정 이다.

◆ 객체에 대하여 참조에 의한 접근이 지원 되지 않는 경우 : 자바에서는 단위 데이터를 제외한 모든 복합 데이터(객체, 배열 등)에 대한 접근 조작을 참조(reference)에 의한 방법으로 처리 한다. 원격 객체에 포함된 멤버 데이터를 참조에 의한 방법으로 접근 조작하기 위해서는 그 객체에 대한 클래스파일 변환이 적용 되어야 한다. 그러나 멤버 데이터로 취급 되는 객체가 JDK에 포함된 라이브러리 클래스이면 다형성이 지원 되지 않는 경우와 동일한 문제점을 야기 한다. 이 경우 에도 새로운 형태의 변환 기법이 적용되어야 하며 향후 적절한 변환 기법을 적용할 예정 이다.

원격 객체에 대하여 현재 언급할 수 있는 서비스의 제한 사항은 이상과 같고, 이러한 제한 사항에도 불구하고 TORB가 지원하는 객체 서비스는 동일한 목적의 기존 시스템 보다 넓은 기능을 제공 한다. 다음의 사실이 이를 뒷 받침 할 수 있다. CORBA의 규약을 따르는 분산 처리 시스템은 IDL을 통하여 인터페이스를 정의한 후, 이를 통한 원격 객체의 접근 조작이 가능한 형태이고, JavaParty는 추가된 키워드 "remote"에 의해 원격 객체로 취급될 객체가 일반적인 객체를 상속 하는 것을 허용 하지 않으므로 TORB의 경우 보다 더욱 좁은 범위의 객체 서비스를 제공 한다.[4,12] HORB는 프로그래밍 투명성을 제공 하지는 않지만 IDL을 적용하지 않고 분산 처리 프로그램을 작성 하는 형태 이며 분산 처리를 지원하기 위하여 상당한 수준의 객체 서비스를 제공 하고 있다.[11] 단위 테스트를 통하여 HORB가 지원 하는 객체 서비스를 모두 포함 하면서 프로그래밍 투명성을 지원 하는 것을 확인함에 따라 TORB가 분산된 컴퓨팅 환경 에서의 객체 서비스를 충분히 지원 한다고 평가할 수 있다.

## Ⅳ. TORB의 성능 평가

TORB는 자바의 병렬 처리 지원 기능을 활용 하여 작성된 레거시 프로그램을 분산된 컴퓨팅 환경을 활용하도록 변환 하고 이를 통한 수행 시간 감소 효과를 제

공 한다. 4장에서는 자바 쓰레드를 활용 하여 작성한 병렬 처리 프로그램에 대한 실험 결과를 분석 하여, TORB가 지원 하는 분산 병렬 컴퓨팅 환경에 대한 성능 평가를 하고자 한다.

### 1. 적용 프로그램

TORB가 지원하는 분산 병렬 컴퓨팅 환경을 평가하기 위해 작성 하는 프로그램은 순회 외판원 문제와 여왕 배치 문제를 구현 한 것이다. 이는 두 가지 모두 문제의 크기가 커짐에 따라 지수 시간이 소요 되는 계산 복잡도를 가지는 것이며, 퇴각 검색법을 적용하여 문제의 해를 구할 수 있는 알고리즘이 존재 하고, 자바 쓰레드를 이용 하여 병렬 처리를 수행 하는 형태로 쉽게 작성 할 수 있기 때문이다. 또한, 각각의 프로그램이 가지는 독특한 동작 특성이 분산된 컴퓨팅 환경 에서의 병렬 동작을 테스트하기에 적당하기 때문 이다. 실험에서는 순회 외판원 문제를 해결 하는 프로그램 두 가지와 여왕 배치 문제를 해결 하는 프로그램 두 가지를 사용 하였다. 순회 외판원 문제는 출발 도시를 제외 하고 제시된 도시의 모든 순열 중 되돌아오는 경로가 가장 짧은 하나의 순열을 선택 하는 것으로서 프로그램 수행 중 쓰레드 상호 간의 정보 교환과 관련 해서 두 가지 형태로 구현 할 수 있다. 하나는 각각의 쓰레드가 독자적인 최적 해를 구하고 마지막에 부모 쓰레드 에게 보고 하는 형태 이고, 다른 하나는 최적 해의 가능성이 있는 순열을 발견 할 때마다 부모 쓰레드 에게 보고 하는 형태 이다. 여왕배치 문제는 정방형의 격자에 규칙을 준수 하며 여왕을 배치하는 모든 경우를 나열 하는 것으로서 순회 외판원 문제 에서와 마찬가지로 쓰레드 상호 간의 정보 교환과 관련해서 두 가지 형태로 구현할 수 있다. 하나는 각각의 쓰레드가 해답을 발견 할 때마다 부모 쓰레드에게 보고 하는 형태 이고, 다른 하나는 발견한 해답을 각자 저장해 두었다가 마지막에 보고 하는 형태 이다. 각각의 프로그램들의 쓰레드 상호간 정보 교환 방식은 독특한 특성을 가지고 있으므로 쓰레드 상호간의 정보 교환 형태와 관련된 분산 병렬 컴퓨팅 환경의 특성을 평가 하는데 적합 하다.

### 2. 프로그램 구현 및 투명성

실험에 적용할 프로그램은 자바의 일반 적인 쓰레드와 동기화 메커니즘을 적용 하여 구현 하였으며 TORB의 후 처리 절차를 적용 하여 변환 한 후 실험 하였다. 이때, TORB가 지원 하는 분산된 컴퓨팅 환경에 적용하기 위한 어떠한 수정도 하지 않았다. 이 사실은 TORB가 제공 하는 프로그래밍 투명성의 범위가 충분히 실효성이 있음을 뒷받침 한다.

### 3. 실험결과 및 평가

실험 에서 관심을 두는 결과는 분산된 컴퓨팅 환경에서의 병렬 처리에 따른 수행 시간 감소 효과의 특성 이다. 이 특성은 쓰레드 상호간 정보 교환의 양과 횟수에 따라 다른 결과를 보일 것이고, 그에 따라 TORB가 지원 하는 분산 병렬 컴퓨팅 환경의 통신 기능과 관련된 오버헤드와 후 처리에 기인한 오버헤드를 평가 할 수 있다.

#### 가. 실험환경

분산 병렬 컴퓨팅 환경에 의한 수행 시간 감소 효과를 실험하기 위하여 100Mbps LAN으로 연결한 네트워크 환경 에서 Windows98이 설치된 컴퓨터(CPU : 펜티엄 IV 2.0GHz, 주 메모리 : 256MB) 14대를 사용 하였으며, TORB가 보유한 자체 오버헤드에 대한 실험을 위해 Windows 2000이 설치된 컴퓨터(CPU : 펜티엄 IV 2.26 GHz, 주 메모리 : 512MB) 1대를 사용 하였다.

#### 나. 수행 시간 감소 효과에 대한 실험 결과 및 평가

순회 외판원 문제에 대한 실험 결과는 표1에 나타난 바와 같으며 총 다섯 가지 프로그램에 대한 수행 시간과 두 가지 종류의 쓰레드 상호간 통신 횟수를 측정 한 결과 이다. 특히, 쓰레드 상호간 통신 횟수는 입력된 자료에 종속적 이므로 각각의 프로그램에는 동일한 자료를 입력 하여 측정 하였다. 표2는 표1에 나타난 다섯 가지 프로그램과 쓰레드 상호간 통신 횟수에 대한 해설 이다.

표 1. 순회 외판원 문제에 대한 실험 결과
Table 1. Examination for the TSP problem.

| PROGRAM Ver. / UNIT / N | TSP-N | TSP-L | TSP-R | TSP-L-ORB | TSP-R-ORB | L-COMM | R-COMM |
|---|---|---|---|---|---|---|---|
| | msec | msec | msec | msec | msec | times | times |
| 2 | 0 | 0 | 0 | 1700 | 1650 | 4 | 4 |
| 3 | 0 | 60 | 0 | 2470 | 2530 | 6 | 8 |
| 4 | 0 | 60 | 60 | 3290 | 4500 | 8 | 30 |
| 5 | 0 | 0 | 60 | 4010 | 9280 | 10 | 88 |
| 6 | 60 | 50 | 60 | 4780 | 11090 | 12 | 172 |
| 7 | 0 | 50 | 50 | 5490 | 7690 | 14 | 71 |
| 8 | 50 | 170 | 50 | 6100 | 12520 | 16 | 99 |
| 9 | 170 | 280 | 220 | 6860 | 13240 | 18 | 129 |
| 10 | 820 | 1100 | 930 | 7580 | 33120 | 20 | 238 |
| 11 | 4340 | 5160 | 4610 | 8840 | 43000 | 22 | 346 |
| 12 | 15210 | 20210 | 16040 | 11590 | 46300 | 24 | 402 |
| 13 | 65960 | 94150 | 75740 | 19990 | 100840 | 26 | 744 |
| 14 | 410240 | 764890 | 656780 | 98670 | 161870 | 28 | 948 |

표 2. 순회 외판원 문제에 적용한 프로그램
Table 2. Comments for the TSP problem.

| | |
|---|---|
| TSP-N | 병렬 처리를 수행 하지 않는 방법으로 구현 한 프로그램 |
| TSP-L | 각각의 쓰레드가 독자적으로 최적해를 구하고, 마지막에 부모 쓰레드 에게 보고 하는 방법 으로 구현한 프로그램 |
| TSP-R | 각각의 쓰레드는 부모 쓰레드로 부터 최신의 최적해를 통보 받고, 더 좋은 해를 구하면 부모 쓰레드 에게 보고 하는 방법으로 구현한 프로그램 |
| TSP-L-ORB | 프로그램 TSP-L에 후 처리를 적용 하여 변환 한 프로그램 |
| TSP-R-ORB | 프로그램 TSP-R에 후 처리를 적용 하여 변환 한 프로그램 |
| L-COMM | 프로그램 TSP-L의 쓰레드 상호간 통신 횟수 |
| R-COMM | 프로그램 TSL-R의 쓰레드 상호간 통신 횟수 |

표 3. 여왕 배치 문제에 대한 실험 결과
Table 3. Examination for the N-Queen problem.

| PROGRAM Ver. | NQ-N | NQ-L | NQ-R | NQ-L-ORB | NQ-R-ORB | L-COMM | R-COMM |
|---|---|---|---|---|---|---|---|
| UNIT / N | msec | msec | msec | msec | msec | times | times |
| 2 | 0 | 0 | 50 | 1260 | 1260 | 2 | 0 |
| 3 | 0 | 0 | 0 | 2200 | 1920 | 3 | 0 |
| 4 | 0 | 0 | 0 | 2800 | 5050 | 4 | 2 |
| 5 | 0 | 0 | 0 | 3240 | 3350 | 5 | 10 |
| 6 | 0 | 60 | 0 | 4010 | 3570 | 6 | 4 |
| 7 | 0 | 0 | 0 | 4170 | 4990 | 7 | 40 |
| 8 | 50 | 50 | 50 | 4940 | 8950 | 8 | 92 |
| 9 | 50 | 60 | 50 | 8070 | 13570 | 9 | 352 |
| 10 | 170 | 170 | 50 | 8560 | 23290 | 10 | 724 |
| 11 | 440 | 390 | 440 | 6260 | 76350 | 11 | 2680 |
| 12 | 1870 | 2200 | 2200 | 7250 | 332960 | 12 | 14200 |
| 13 | 11590 | 12740 | 13010 | 9780 | 1788390 | 13 | 73712 |
| 14 | 78160 | 80020 | 85410 | 45860 | 8870037 | 14 | 365596 |

순회 외판원 문제에 대한 실험 결과에서 TSP-R이 TSP-L 보다 수행 시간이 더 작은 것을 확인 할 수 있는데 이는 쓰레드간의 정보 교환에 의해서 퇴각 검색법의 효과를 높인 결과 이다. 그러나 분산 병렬 컴퓨팅 환경에서 동작 하는 TSP-R-ORB는 통신의 오버헤드 때문에 TSP-L-ORB 보다 수행 시간이 더 커진다. 분산 병렬 컴퓨팅 환경에 의한 수행 시간 감소의 효과는 TSP-L-ORB에서는 입력된 도시의 개수가 12인 경우부터, TSP-R-ORB에서는 14인 경우부터 나타났다. 특히, 쓰레드 상호간 통신 횟수가 가장 작은 TSP-R-ORB의 경우 수행 시간 감소 효과가 나타나기 시작 할 때부터의 수행 시간 감소 비율은 1.3213, 3.2996, 4.2437 (15210/11590, 65960/19990, 410240/96670)로 점점 증가함을 보이고 있으며 궁극적으로는 병렬 처리에 참여 하는 컴퓨터(동일한 컴퓨팅 능력을 가진)의 수에 정 비례 할 것이라는 예측을 할 수 있다.

여왕 배치 문제에 대한 실험 결과는 표 3에 나타난 바와 같으며 총 다섯 가지 프로그램에 대한 수행 시간과 두 가지 종류의 쓰레드 상호간 통신 횟수를 측정한 결과 이다. 표4는 표3에 나타난 다섯 가지 프로그램과 쓰레드 상호간 통신 횟수에 대한 해설 이다. 여왕 배치 문제에 대한 실험 결과에서는 순회 외판원 문제에서와

표 4. 여왕 배치 문제에 적용한 프로그램
Table 4. Comments for the N-Queen problem.

| | |
|---|---|
| NQ-N | 병렬 처리를 수행 하지 않는 방법으로 구현 한 프로그램 |
| NQ-L | 각각의 쓰레드가 구한 해를 각자 보관 해 두었다가 마지막에 부모 쓰레드 에게 보고 하는 방법으로 구현한 프로그램 |
| NQ-R | 각각의 쓰레드가 해를 구할 때 마다 부모 쓰레드 에게 보고 하는 방법으로 구현 한 프로그램 |
| NQ-L-ORB | 프로그램 NQ-L에 후 처리를 적용 하여 변환 한 프로그램 |
| NQ-R-ORB | 프로그램 NQ-R에 후 처리를 적용 하여 변환 한 프로그램 |
| L-COMM | 프로그램 NQ-L의 쓰레드 상호간 통신 횟수 |
| R-COMM | 프로그램 NQ-R의 쓰레드 상호간 통신 횟수<br>이는 여왕 배치 문제에 대한 해답의 개수와 같다. |

는 달리 NQ-L과 NQ-R의 수행 시간은 큰 차이를 보이지 않는다. 이는 NQ-R에서 수행 하는 쓰레드간의 정보 교환이 퇴각 검색법의 효과와는 관련이 없기 때문이다. 오히려, 통신 횟수가 가장 많은 NQ-R-ORB에서 오버헤드의 결과만 나타난다.

NQ-L-ORB와 NQ-R-ORB는 쓰레드간의 통신에 의해 교환할 정보의 양은 동일하고 통신 횟수는 현격한 차이가 있다. 따라서 NQ-R-ORB을 통한 수행 시간 감소의 효과는 기대 하기 어렵고, NQ-L-ORB에서는 배치할 여왕의 수가 14일 때 수행 시간 감소의 효과가 나타나기 시작 한다. 이는 순회 외판원 문제의 TSP-L-ORB에서와 비슷한 수행 시간에서 나타나는 효과 이며, NQ-L-ORB와 NQ-R-ORB의 쓰레드 상호간 정보 교환의 양이 동일 하다는 것을 고려 하면 TORB의 통신과 관련된 오버헤드는 통신 횟수에 더 많이 관여 된다는 판단을 할 수 있다.

두 가지 문제에 대한 실험 결과에서, TORB가 지원하는 분산 병렬 컴퓨팅 환경을 이용 하면, 프로세스 간 통신 횟수가 적은 병렬 프로그램을 자바의 쓰레드를 이용하여 쉽게 작성할 수 있고, 병렬 처리에 의한 수행시간 감소 효과를 기대할 수 있음을 확인 할 수 있다.

### 다. TORB의 오버헤드에 대한 실험결과 및 평가

표 5에는 TORB의 후처리에 의해 변환된 코드가 지니는 오버헤드를 예측하기 위한 실험 결과가 나타나 있다. 이 실험은 한 대의 컴퓨터에서 서로 다른 URL을 지정한 14개의 TORB 서버를 기동 한 후, 이를 통한 분산 병렬 처리를 수행한 결과 이다. 이렇게 하면, 통신 동작이 루프 백에 의해 이루어지므로 쓰레드 상호간 정보 교환 속도를 최대화 하여 통신 속도에 의한 오버헤드는 무시할 수 있고, 후처리에 의해 변환된 코드가 지니는 오버헤드(운영체제의 통신 드라이버가 지니는 오버헤드가 포함된 것이므로 전적으로 후처리에 의한 오

표 5. TORB의 오버헤드 평가를 위한 실험 결과
Table 5. Examination for extimate overhead.

| PROGRAM Ver. | NQ-N | NQ-L | NQ-R | NQ-L-ORB | NQ-R-ORB | L-COMM | R-COMM |
|---|---|---|---|---|---|---|---|
| UNIT / N | msec | msec | msec | msec | msec | times | times |
| 2 | 0 | 16 | 15 | 188 | 171 | 2 | 0 |
| 3 | 0 | 16 | 16 | 250 | 344 | 3 | 0 |
| 4 | 0 | 15 | 16 | 344 | 422 | 4 | 2 |
| 5 | 0 | 16 | 15 | 375 | 344 | 5 | 10 |
| 6 | 0 | 15 | 16 | 391 | 610 | 6 | 4 |
| 7 | 0 | 16 | 15 | 469 | 734 | 7 | 40 |
| 8 | 0 | 31 | 16 | 516 | 1015 | 8 | 92 |
| 9 | 16 | 16 | 16 | 766 | 2219 | 9 | 352 |
| 10 | 47 | 62 | 62 | 734 | 3859 | 10 | 724 |
| 11 | 265 | 282 | 281 | 1312 | 11344 | 11 | 2680 |
| 12 | 1609 | 1703 | 1610 | 3281 | 52063 | 12 | 14200 |
| 13 | 10266 | 10219 | 10422 | 14031 | 246437 | 13 | 73712 |
| 14 | 69437 | 68781 | 69969 | 82594 | 1200813 | 14 | 365596 |

버헤드는 아님)를 예측할 수 있다.

실험에 사용된 프로그램은 여왕 배치 문제를 적용 하였다. 이는 쓰레드 상호간 정보 교환 횟수가 일정 하고, NQ-L-ORB와 NQ-R-ORB간의 정보 교환 횟수가 현저하게 차이 나는 특성을 포함 하고 있기 때문이다. 한 대의 컴퓨터에서 분산 병렬 컴퓨팅 환경을 구성 하여 수행한 결과 이므로 수행 시간 감소 효과는 당연히 나타나지 않고 오히려 수행 시간이 증가한 결과를 보인다. NQ-N와 NQ-N-ORB를 비교해 보면 N이 9인 경우 부터 수행 시간의 증가 비율이 47.87, 15.62, 4.95, 2.04, 1.36, 1.19 (766/16, 734/47, 1312/265, 3281/1609, 14031/10266, 82594/69437)로 총 수행 시간이 증가함에 따라 점점 감소함을 확인 할 수 있다. 특히, NQ-N의 총 수행 시간이 70초 정도일 때, NQ-N-OR B는 1.19배 증가한 83초 정도를 보이고 있다. 이는 한 대의 컴퓨터에서 동시에 동작 하는 14개의 쓰레드가 보유한 오버헤드를 합친 것 이므로 실제의 오버헤드는 1+0.19/14로서 1.0136배 정도 이다. 따라서 TORB에 의해 변환된 코드가 가지는 오버헤드가 실제 응용 프로그램에 적용 하여도 수용 가능 하다고 평가 할 수 있다.

### 4. TORB와 병렬 처리 지원 환경

일반적으로 병렬 처리를 수행 하는 프로그램은 하나의 실행 공간 내에서 두 개 이상의 프로세스를 동시에 수행 할 수 있도록 작성 하며 적절한 병렬 처리 환경에서 동작 한다. 이를 위해서는 동시에 수행 하는 여러 개의 프로세스를 다룰 수 있는 방안이 프로그래밍 언어에 마련되어 있어야 하고, 작성된 프로그램이 병렬 처리를 수행 하며 동작 할 수 있는 환경이 필요 하다. 병렬 처리와 관련된 프로그래밍 언어의 지원은 라이브러리 함수를 제공 하는 것으로 기존 언어의 기능을 확장하는 방안과 프로그래밍 언어 자체의 기능 으로서 제공하는 방안이 있다. 특히, 자바는 쓰레드를 미리 정의 되고 다루기 쉬운 객체 형태로 제공 하고, 동기화 메커니즘을 프로그래밍 언어 자체의 기능으로 제공 하고 있으므로, 자바가 제공 하는 쓰레드와 동기화 메커니즘은 병행 처리를 수행 하는 프로그램을 쉽게 작성 하는데 상당한 역할을 담당 한다.

병렬 처리를 수행하는 프로그램이 동작할 환경은 오늘날의 다중 처리 환경 이면 충분 하나 병렬 처리를 통한 수행 시간 감소 효과를 얻기 위해서는 여러 개의 프로세서가 장착된 시스템이나 분산 되어 있는 컴퓨팅 자원을 통합 하여 활용할 환경이 필요 하다. 병렬 처리를 지원하기 위한 방안으로서 분산된 컴퓨팅 환경을 활용하는 것은 새로운 패러다임을 제공 하는 의미를 가지므로 최근에 많은 연구의 대상이 되고 있다.[2,3,4,5,6]

2장에서 제시한 바와 같이, 분산된 컴퓨팅 환경을 병렬 처리를 지원 하는데 활용하기 위해서, TORB는 프로그래밍 투명성을 지원함과 동시에 자바의 실행 환경에 대한 범용성 및 투명성을 유지 하는 기능을 제공 하여 기존의 솔루션을 개선하였다. 4장의 실험 결과를 통하여, TORB가 분산된 컴퓨팅 환경을 병렬 처리를 위한 자원으로 활용 하는 방안을 제공 하며 개선된 기능을 동시에 지원함을 입증 하였고, TORBC에 의해 변환된 코드가 지니는 오버헤드는 TORB의 실용성을 제한하지 않음을 입증 하였다. 쓰레드 상호간의 정보 교환과 관련된 오버헤드는 병렬 처리 시스템이 가지는 본질적인 문제를 그대로 반영 하는 것이므로 새삼스러운 것이 아니다.

## V. 결론 및 향후연구

자바 쓰레드를 적용 하여 병렬 알고리즘을 구현 한 프로그램을 여러 대의 컴퓨터가 협력 하여 수행 하는 분산 처리 환경 에서 수행 한다면 쓰레드의 동시 수행에 의해 수행 시간 단축의 효과를 얻을 수 있다. 또한, 이러한 프로그램을 분산 처리 환경에 대한 부가적인 지식 없이 작성 할 수 있다면, 주어진 컴퓨팅 환경에 적용하기 위한 프로그램 개발자의 부담을 줄여서, 개발하는 소프트웨어의 자체 기능에 더욱 집중 할 수 있다. 본 논문에서는 프로그래밍 투명성에 의하여 분산된 컴퓨팅 환경을 전혀 고려하지 않고 작성된 병렬 처리 프로그램을 후 처리 과정을 거쳐서 여러 대의 컴퓨터로 구성된 분산 병렬 컴퓨팅 환경 에서 수행 하고 수행 시간 단축

효과를 얻을 수 있는 컴퓨팅 환경을 제안 하고, 이름을 TORB라고 명명 하였다. TORB는 표준 자바 가상 기계에서 동작 하는 서드파티 패키지와 보조 프로그램들로 구성 되어 있으므로 자바의 범용성을 분산된 컴퓨팅 환경에서도 그대로 유지할 수 있다. 일반적으로 수행 시간이 오래 걸리는 문제로 알려진 순회 외판원 문제와 여왕 배치 문제를 통하여 실험 결과를 보였으며, 이를 통하여 수행시간 단축의 효과가 나타난 결과를 제시 하였다. 프로그래밍 투명성, 자바의 범용성 유지, 단순한 후처리 과정, 사용상의 용이성은 본 논문에서 주장 하는 중요 사항이며, 이를 통하여 분산된 컴퓨팅 환경을 활용 하는 응용 소프트웨어의 개발이 용이해 질 수 있다. 본 논문에서 주장하는 프로그래밍 투명성은 별도의 정형화된 연구가 필요 할 것이며, 완전한 프로그래밍 투명성 및 보안 기능을 구현 하기 위한 지속적인 기능 개선이 필요 하다.

## 참 고 문 헌


[1] Doug Lea, "Concurrent Programming in Java - Design Principles and Patterns", Addison -Wesley, Reading, Mass., 1996.

[2] Weimin Yu and Alan Cox, "Java/DSM: A Platform for Heterogeneous Computing", In Concurrency: Practice & Experience, Vol. 9, No. 11, pp. 1213-1224, 1997.

[3] Yariv Aridor, Michael Factor and Avi Teperman, "cJVM: a Single System Image of a JVM on a Cluster", In Proceedings of ICPP 99, IEEE, 1999.

[4] Philippsen M, Zenger M. "JavaParty - Transpa -rent remote objects in Java", Concurrency: Practice and Experience, 9(11):1225--1242, 1997.

[5] M. Philippsen, B. Haumacher, and C. Nester, "More efficient serialization and RMI for Java", Concurrency: Practice & Experience, 12(7):495 -518, May 2000.

[6] Matchy J. M. Ma, Cho-Li Wang and Francis C. M. Lau, "JESSICA: Java-Enabled Single-Sys tem-Image Computing Architecture", In Journal of Parallel and Distributed Computing, Vol. 60, No. 11, pp.1194-1222, 2000.

[7] 이상윤, 김승호, "프로그래밍 투명성을 지원하는 분산 프로그래밍 도구의 설계", 한국정보과학회 논문집 제 31권 3호, pp. 259-268, June 2004.

[8] S. Maffeis, "Run-Time Support for Object -Oriented Distributed Programming", PhD thesis, University of Zurich, 1995.

[9] Forum for Gnutella Development, "Gnutella.com". Online Document, http://www.gnutella.com/

[10] Sun Microsystems, Inc., "Java Remote Method Invocation", Online publishing, URL http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html, 2003.

[11] Satoshi Hirano, "HORB : Distributed Execution of Java Programs", In Lecture Notes in Computer Science 1274, Springer, pp.29-42, 1997.

[12] Bernhard Haumacher, Thomas Moschny, Jurgen Reuter and Walter F. Tichy, "Transparent Distri -buted Threads for Java", International Parallel and Distributed Processing Symposium, 2003.

[13] J. Siegel, "CORBA Fundamentals and Program -ming", John Wiley & Sons, 1996.

[14] M. Dahm. "Byte code engineering with the BCEL API". Technical Report B-17-98, Freie Universitat Berlin, Institut fur Informatik, April 2001.

[15] J. Meyer, T. Downing. "Java Virtual Machine". O Reilly, 1997.


## 저 자 소 개


이 상 윤(정회원)

1993년 경북대학교 컴퓨터 공학과 졸업(공학사).

1995년 경북대학교 대학원 컴퓨터 공학과 졸업(공학석사).

2000년 8월 경북대학교 대학원 컴퓨터공학과(박사수료).

1997년~현재 대원과학대학 컴퓨터정보처리과 조교수.

<주관심분야: 분산 시스템, 그리드 컴퓨팅, 멀티미디어, 자바 응용기술, 임베디드 시스템 등>

김 승 호(정회원)

제 40권 TC편 제 7호, 참조