병렬 객체지향 프로그래밍을 위한 시각 환경의 설계 및 구현

최 숙 영†

요 약

병렬 프로그래밍은 프로세스간의 통신과 동기화 문제, 병렬 시스템의 구성 형태 등을 고려해야 하기 때문에 순차 프로그래밍에 비해 많은 노력을 필요로 한다. 효율적인 병렬 프로그램을 작성하기 위해서는 사용자와 컴파일러간의 상호 지원이 이루어져야 한다. 이러한 관점에서 본 연구는 선행 연구로써 병렬 객체지향 표기언어 POOSL을 개발하였다. 그러나, 사용자 입장에서 볼 때 병렬 프로그램을 작성하기 위해 POOSL의 문법 구조를 염두에 두고 텍스트 중심의 프로그램을 작성한다면 여전히 부담스러운 작업이 될 것이다. 사용자에게 보다 편리함을 제공하기 위해서는 텍스트보다는 시각적인 프로그래밍 환경이 더욱 효율적이고 바람직할 것이다. 따라서, 본 논문에서는 POOSL을 기초로 하여 사용자가 좀더 쉽고, 편리하게 병렬 프로그래밍 할 수 있는 시각 환경으로써 VEPO(Visual Environment for Parallel Object-Oriented Programing)를 제안하고 있다. 본 논문의 목적은 사용자가 병렬 프로그램을 작성하는데 있어 문제에 내재된 병렬성을 객체지향 개념에 입각하여 시각적으로 자연스럽게 표현하도록 하고, 병렬 프로그램 개발에 관련된 과정들을 하나의 환경으로 통합시킴으로써 편리한 프로그램 환경을 제공하는 것이다. 본 연구에서 제안하고 있는 VEPO는 병렬 프로그램을 개발하는데 필요한 기본적인 단계들로써 프로그램 기술 단계, 실행 단계, 실행 과정의 시각화등을 지원하고 있으며, 시각 프로그래밍의 장점을 충분히 살릴 수 있도록 여러 개념들이 지원되고 있다. 특히, 병렬 프로그램에서 복잡하고 까다로운 통신과 동기화에 관련된 코드 등은 번역 과정에서 자동적으로 생성되도록 함으로써 사용자로 하여금 병렬 프로그램을 작성하는데 따르는 부담감을 줄일 수 있도록 한다. 본 시스템은 PC를 호스트로 연결한 트랜스퓨터들로 구성된 병렬 컴퓨터 MC-3에서 구현되었다. VEPO 그래픽 사용자 인터페이스는 Visual C++로 구현되었고, VEPO에서 작성된 시각 프로그램은 Inmos C 코드로 번역되어 MC-3에서 수행된다.

Design and Implementation of a Visual Environment for Parallel Object-Oriented Programming

Sook-Young Choi†

ABSTRACT

Comparing with sequential programming, parallel programming has additional complexity due to the consideration of parallelism, communication and synchronization of processes. A synergism between users and compilers should be established, each assisting the other to produce high quality parallel programs. On the above underlying philosophy, we developed a parallel Object-Oriented Specification Language, POOSL, as preliminary works. However, it is still likely to hard for users to write parallel program because users have to consider grammar of POOSL and to write text-based parallel program. It would be more desirable to provide users with visual environment for effective parallel programming. Therefore, we propose a visual programming environment, VEPO(Visual Environment for Parallel Object-Oriented Pro-

* 본 연구는 1998학년도 우석대학교 연구 지원비에 의하여 연구되었음.
† 정 회 원 : 우석대학교 컴퓨터교육과 교수
논문접수 : 1998년 7월 28일, 심사완료 : 1998년 11월 21일

gramming), based on POOSL in order that users can develop parallel programs more easily and conveniently. It aims at supporting a programming environment in which users can represent their programs more naturally and visually in parallel manner with object-oriented concept and essential steps during parallel program development such as program specification, compilation, execution and animation of execution are integrated. VEPO has useful features for parallel processing. Especially, complicated parallel codes for synchronization and communication of processes are automatically generated in the translation phase, so users can be relieved of writing error-prone parallel codes. The system is targeted to the transputer-based parallel system, MC-3. The graphic user interface of VEPO was implemented using Visual C++. Visual programs described on VEPO are translated into Inmos C and executed on MC-3.

1. 서 론

병렬 프로그래밍은 프로세스간의 통신과 동기화 문제, 병렬 시스템의 구성 형태 등을 고려해야 하기 때문에 순차 프로그래밍에 비해 많은 노력을 필요로 한다. 이러한 병렬 프로그래밍을 위한 연구는 크게 두 가지 관점에서 접근되고 있다[1]. 먼저 기존의 순차 언어로 작성된 프로그램을 병렬 프로그램으로 바꾸어 주는 병렬화 컴파일러(parallelizing compiler)를 개발하는 방법과, 직접 병렬처리에 적합한 병렬 프로그래밍 언어를 이용하여 직접 사용자가 병렬 프로그램을 작성하는 방법이다. 그러나 이 두 가지 방법들은 각각 장·단점을 가지고 있기 때문에 어느 한가지 방법만을 가지고 병렬 프로그램을 수행하기에는 무리가 따른다. 따라서 효율적인 병렬 프로그램을 작성하기 위해서는 사용자의 노력과 컴파일러의 지원이 상호 이루어져야 한다. 이러한 관점에서 본 연구는 선행 연구로써 병렬 객체지향 표기언어 POOSL을 개발하였다[2,3]. POOSL은 사용자가 병렬 프로그램을 쉽게 작성하기 위해 실행에 관련된 자세한 코드까지 고려할 필요없이 병렬성에 관한 고수준의 구문을 사용하여 주어진 문제를 효과적으로 표현할 수 있는 기술언어를 개발하는 것을 목적으로 하였다. 특히, POOSL은 객체지향 모델에 바탕을 두고 있다. 객체지향 모델에서 객체는 하나의 독립적인 모듈로서, 객체들간의 정보 교환은 메시지 전송을 통하여 수행되는데, 이러한 수행 방식은 병렬처리와 연관지어 설명될 수 있다. 즉, 객체지향 개념에서 객체를 병렬처리 시스템의 프로세스라는 개념으로, 객체간의 전송은 병렬처리 시스템에서 프로세스사이의 통신 방식으로 자연스럽게 대응시킬 수 있다. 또한, 객체지향 모델은 현실 세계를 자연스럽게 반영할 수 있고, 구현 단계에서 보다는 소프트웨어 개발의 초기 단계에서 객체의 개념이 자연스럽게 도입될 수 있기 때문에 병렬성 및 통신과 같은 병렬처리의 관점을 소프트웨어의 초기 단계에서 사용자가 자연스럽고 쉽게 다룰 수 있다. 따라서, 이러한 객체지향 개념에 바탕을 두고, 병렬성을 효과적으로 수행하려는 연구들로써 병렬 객체지향 언어들이 개발되었다[1,4,5].

그러나, 사용자 입장에서 볼 때 병렬 프로그램을 작성하기 위해 POOSL의 문법 구조를 염두에 두고 텍스트 중심의 프로그램을 작성한다면 여전히 부담스러운 작업이 될 것이다. 그러므로, 사용자에게 보다 편리함을 제공하기 위해서는 텍스트보다는 시각적인 프로그래밍 환경이 더욱 효율적이고 바람직할 것이다. 시각적인 프로그래밍 방법은 많은 장점을 가지고 있다[6,7]. 첫째, 사용자가 프로그램의 구성을 쉽게 표현할 수 있고 변경할 수 있다. 둘째, 프로그램의 구조를 쉽게 파악할 수 있다. 셋째, 표현된 시각 프로그램 자체를 프로그램의 수행 과정을 검증하고, 성능 분석을 하는데 이용할 수 있다.

따라서, 본 연구는 POOSL을 기초로 하여 사용자가 좀더 쉽고, 편리하게 프로그래밍 할 수 있는 병렬 시각 프로그래밍 환경으로서 VEPO(Visual Environment for Parallel Object-Oriented Programming)를 제안하고 있다. 본 논문의 목적은 사용자가 병렬 프로그램을 작성하는데 있어 문제에 내재된 병렬성을 객체지향 개념에 입각하여 시각적으로 자연스럽게 표현하도록 하고, 병렬 프로그램 개발에 관련된 과정들을 하나의 환경으로 통합시킴으로써 편리한 프로그램 환경을 제공하는 것이다. 본 연구에서 제안하고 있는 VEPO는 객체지향 개념에 입각하여 병렬 프로그램의 프로세스에 대응되는 객체노드 아이콘과 프로세스간의 통신에 대응되는 객체 노드들간의 간선 아이콘을 이용하여 병렬 프로그램을 시각적으로 표현하도록 하기 때문에 문제

에 내재된 병렬성을 자연스럽고 쉽게 표현할 수 있다. 이러한 객체단위의 병렬성 뿐만 아니라, 객체내 메소드간의 병렬성과 자료 병렬성을 지원하고 있다. 또한 병렬 프로그램을 개발하는데 필요한 기본적인 단계들로써 프로그램 기술 단계, 번역 단계, 실행 단계, 실행 과정의 시각화 등을 하나의 통합 환경으로 지원함으로써 좀더 효과적으로 병렬처리를 수행할 수 있게 한다. 작성된 시각 프로그램 자체를 이용하여 프로그램의 실행 과정을 시각적으로 보여줄 수 있다.

본 시스템은 PC를 호스트로 연결한 트랜스퓨터로 구성된 병렬 컴퓨터 MC-3에서 구현되었다. VEPO에서 작성된 시각 프로그램은 먼저 POOSL 코드로 번역된 후, 트랜스퓨터에서 수행되는 병렬 C인 Inmos C 코드로 번역되어 MC-3에서 수행된다. 이러한 번역 과정중에 병렬 프로그램의 복잡하고 까다로운 부분인 통신과 동기화에 관련된 코드가 자동적으로 생성됨으로서 사용자는 병렬 프로그램을 작성하는데 따르는 부담감을 줄일 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로써 대표적인 병렬 시각 프로그래밍 방법들의 특징들과 본 연구와의 차이점을 살펴보고, 3장에서는 VEPO의 기본 특징 및 프로그램 개발 환경 등을 설명한다. 4장에서는 VEPO의 전반적인 시스템 구성과 구현에 대해 설명한다. 끝으로, 결론 및 앞으로의 연구 방향을 5장에서 제시한다.

2. 관련 연구

시각 프로그래밍을 위한 연구들로써 국내에서는 경북대학교에서 시각 언어에 대해 연구하였으며[8], 외국에서는 Glinert가 대화식 그래픽 프로그래밍 환경인 Pict라는 시스템[9]을, Hiraeawa가 아이콘 프로그래밍 시스템인 HI-VISUAL[10]을 개발하였다. 그러나 이들 연구들은 모두 순차 프로그래밍 언어에 기반을 두고 개발된 시각 프로그래밍 환경이다.

병렬 시각 프로그래밍을 위한 연구들로써 Tennessee 대학에서 개발한 VPE[11]가 있다. VPE는 메시지 전달(mesage-passing) 방법에 기초한 프로그래밍 기법으로써 사용자가 쉽게 병렬프로그램을 작성할 수 있도록 간단한 인터페이스를 제공하고 있다. 또한 VPE는 PVM과 같은 일반적인 메시지 패싱 라이브러리들을 사용할 수 있게 함으로써 이종의 컴퓨터간에도 수행할 수 있다. 그러나, VPE는 계산 단위(computation unit) 간에 수행되는 통신을 표현하기 위해 입·출력 포트(port)를 사용자가 일일히 지정해야 하는 불편함이 존재하며, 병렬성을 지원하는 부분도 제한적이다.

또한 이와 유사한 연구들로써 Texas 대학과 Tennesse 대학의 공동연구로 개발한 CODE[12,13]와 HeNCE[14,15]가 있다. 이들 연구들은 위에서 살펴본 VPE의 명시적인 메시지 전달 모델에 기초하지 않고, 좀더 고수준의 통신 메카니즘을 사용하고 있다. 이 두 모델의 프로그램 과정은 크게 두과정으로 구분되고 있다. 먼저, 주어진 문제에서 순차적인 계산 단위들을 고려한 후 정의한다. 다시 이 순차적 모듈들은 병렬 프로그램을 구성하기 위해 하나의 그래프에 재배치된다. 각 순차적 모듈간의 통신은 모듈을 수행하는 동안에 발생되지 않고, 모듈의 시작이나 끝 부분에서 수행된다.

HeNCE와 CODE는 위와 같이 서로 유사한 방법을 추구하고 있지만, 기본적으로 가정하고 있는 모델이 다르다. 먼저 CODE 모델은 데이터 플로우 모델에 기초하고 있으며, HeNCE는 구조적 병렬 순서도 방식(structured parallel flowchart)에 기본을 두고 있다. 이들 방법은 병렬 프로그램 작성시 내부적인 수행 과정을 자세히 고려할 필요 없이 병렬로 수행될 수 있는 순차 모듈들을 그래프로 표현하기 때문에 보다 쉽게 프로그램을 작성할 수 있다는 장점이 있다. 그러나 처리하는 작업에 따라서, 한 프로세스가 순차모듈을 수행 중에 다른 프로세스와 통신해야 할 일들이 많이 발생될 경우에는 적합하지 못하다.

본 연구에서 제안한 VEPO는 객체지향에 기초하여 병렬 프로그램에서의 프로세스들을 객체들로 프로세스간의 통신 방법을 객체간의 메시지 전달 방식으로 표현하고 있기 때문에 CODE와 HeNCE에서와 같은 문제점이 발생되지 않고, 사용자 입장에서 좀더 자연스럽게 병렬 프로그램을 작성할 수 있다. 또한 VEPO는 고성능(high-performance) 계산에 적합한 자료 병렬성(data parallelism)을 지원하고 있기 때문에 효과적인 병렬 처리를 수행할 수 있다.

3. VEPO(Visual Environment for Parallel Object-Oriented Programing)

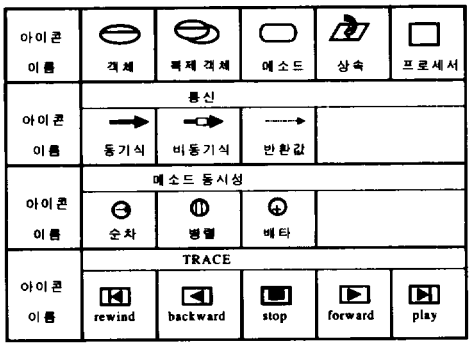
VEPO는 병렬 시각 프로그래밍 환경으로써, 객체지향 모델에 기초하고 있다. 실세계는 병행성의 특징을

가지며, 객체지향 모델은 이러한 실세계를 자연스럽게 반영할 수 있다. 객체지향 모델은 모듈간의 연관성이 적으며, 객체간의 대화 수단으로 메시지 전송을 사용하기 때문에 병렬처리로 확장이 용이하다. VEPO는 이러한 객체지향 모델이 갖는 기본적인 병렬성 위에 여러 병렬성을 위한 개념들이 지원되고 있다. 병렬 프로그램을 개발하는데 VEPO에서 지원하고 있는 유용한 특징들은 다음과 같다.

- 프로그램을 작성하기 쉽고, 작성된 프로그램을 이해하기 쉽다.
- 작성된 프로그램을 이용하여 프로그램의 수행 과정을 시각화한다.
- 문제의 내재된 병렬성을 자연스럽게 표현할 수 있다.
- 한 객체안에 존재하는 메소드간의 병렬성을 표현할 수 있으며, 고성능 계산에 적합한 자료 병렬성도 지원되고 있다.
- 병렬 프로그램 개발에 관련되는 여러 과정들을 하나로 통합한 프로그래밍 환경을 지원한다.
- 번역 과정 중에 프로세스간의 통신과 동기화에 관련된 코드들이 자동적으로 생성된다.

3.1 구성 요소

VEPO의 구성 요소들은 POOSL에서 제공하고 있는 개념들에 기초하고 있으며, 이와 관련된 아이콘들이 (그림 1)에 정리되어 있다.



(그림 1) VEPO의 구성 아이콘들
(Fig. 1) Component Icons of VEPO

3.1.1 객체

프로그램의 기본 단위로 한 계산 단위를 나타낸다. 각 객체는 이름이 부여되고, 연산을 위한 메소드들을 포함한다. 객체에는 활성(active) 객체와 수동(passive) 객체가 있으며, 활성 객체는 자신의 제어를 가지고 있으며 생성시 활성화되며, 수동 객체는 다른 객체로부터 메시지를 받고 활성화된다. 한 객체는 복제(replication) 형태를 가질 수 있으며, 이를 위해 (그림 1)에서 볼 수 있는 복제 아이콘을 사용하여 생성될 수 있다. 이 경우 지정된 갯수만큼의 동일 객체들이 동시에 수행되며, 이것은 자료 병렬성의 문제를 기술하는데 사용된다.

3.1.2 통신

객체 노드간에는 메시지를 전달함으로써 통신이 이루어지며, 이 통신은 단방향 간선(uni-direction arc)을 이용하여 표현된다. 통신 방식에는 동기식·비동기식과 future variable을 이용한 미래형 통신 방식이 지원되며, 이들 각각에 대한 표현 방법은 (그림 1)에서 볼 수 있는 바와 같이 각각 다르게 표현되며, 각 방식에 대한 자세한 설명은 [2]에서 참조될 수 있다.

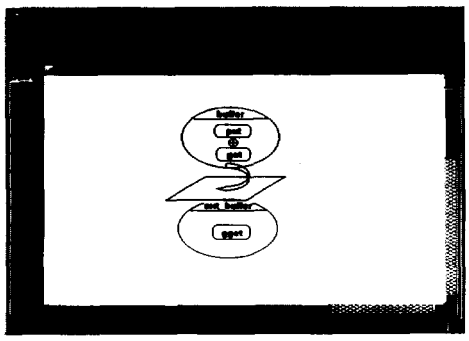
3.1.3 상속성

VEPO는 상속성의 개념을 지원한다. 한 객체는 상위 객체로부터 상속되어질 수 있다. 이때 상위 객체의 메소드들이 하위 객체에 상속되어 재사용될 수 있으며, 그 메소드들의 동기화 코드도 부분적으로 상속되어 질 수 있다. (그림 2)는 상위 객체 buffer 객체로부터 상속되고 새로운 메소드 gget이 추가된 하위 객체 ext-buffer 객체를 보여준다. Buffer 객체에 대한 설명은 3.2절에 기술된다.

3.1.4 Intra-object 동시성

VEPO에서는 한 객체안에 존재하는 메소드들의 동시성(concurrency)과 수행 규약(constraint)을 시각적으로 표현할 수 있다. 한 객체에 정의되는 메소드들 중에는 동시에 수행 가능한 것들과 동시에 수행될 수 없는 것들이 존재할 수 있다. 이러한 메소드들의 관련성을 시각적으로 표현함으로써 한 객체안의 병렬성을 쉽게 파악할 수 있다. 또한 이 정보들은 상속성 이상(inheritance anomaly) 문제를 해결하는데 중요한 역할을 한다[15]. 만약에 한 객체에 있는 메소드들간의 수행규약이 명시적으로 표현되지 않으면, 그 메소드들은 서로 배타적(exclusive)으로 수행된다고 가정한다. (그

림 2)의 buffer 객체에서 메소드 put과 get은 동시에 수행될 수 없다는 것을 의미한다.



(그림 2) Extended-bounded buffer
(Fig. 2) Extended-bounded buffer

3.1.5 자료 병렬성

자료 병렬성은 각 프로세스들이 서로 다른 데이터를 가지고 동일 연산을 동시에 수행하는 것이다. 이러한 자료 병렬성을 위한 프로그램은 크게 호스트(host) 모듈과 노드(node) 모듈들로 구성된다. 호스트 모듈은 각 노드들(가상 프로세서들)에 자료를 분배하는 부분이 정의되며, 노드 모듈은 분배받은 데이터를 가지고 가상 프로세서상에서 수행할 동일 프로시쥬어(procedure)를 기술한다. 호스트 모듈의 자료 분배 코드는 그래픽 작업 창(graphic working window)에서 각 객체들을 표현한 후, 부가 정보 창(embedded information window)을 호출하여 data distribution 필드에 표현한다. 노드 모듈을 정의할 때에는 복제 객체 아이콘을 이용하여 기술한다. 이 자료 병렬성에 대한 예제 프로그램은 3.3.2절에서 보여진다.

3.2 VEPO 그래픽 사용자 인터페이스(VEPO Grapic User Interface)

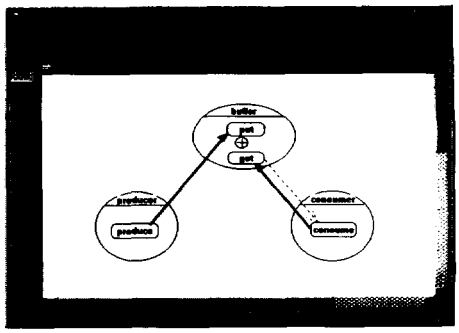
VEPO 그래픽 사용자 인터페이스로써 병렬 프로그램을 시각적으로 구성하기 위한 그래픽 작업 창과 실행에 필요한 정보를 기술하기 위한 부가 정보 창이 존재한다.

3.2.1 그래픽 작업 창

1) 프로그램 구성

프로그램의 각 계산 단위들을 구성하고 작성하는 기본 창으로 기존의 다른 윈도우 시스템의 작업 환경과 유사하다. (그림 3)에서 볼 수 있는 것과 같이, 창의 상단에서 주메뉴들이 존재하고, 창의 왼쪽에는 프로그램을 구성하는 아이콘들이 있다. 새로운 프로그램을 작성하기 위해, FILE 메뉴판의 NEW를 선택한 후 해당 아이콘들을 클릭한 다음, 오른쪽 작업 화면에서 모듈들을 구성한다. 하나의 프로그램은 여러개의 그래픽 프로그램 화일들로 구성될 수 있다. PROJ 메뉴를 선택하면 프로젝트 창이 생성되는데, 이때 프로젝트 화일명과 그 프로젝트 화일을 구성하는 각 그래픽 프로그램 화일명을 지정한다.

(그림 3)은 그래픽 작업 창에서 작성된 하나의 프로그램으로써 bounded-buffer에 관한 프로그램이다. 이 프로그램은 세개의 객체들로 구성되며, producer 객체와 consumer 객체는 활성 객체로써 원모양의 아이콘을 사용하여 구성하였으며, buffer 객체는 수동 객체로써 타원 모양의 아이콘으로 정의되었다. Producer 객체는 버퍼에 데이터 아이템(item)을 집어넣고, consumer 객체는 버퍼로부터 데이터 아이템을 꺼내는 객체이다. 이 producer 객체와 consumer 객체들은 각각의 일을 수행하기 위해 buffer 객체의 put과 get 메소드를 호출하고 있다.



(그림 3) Bounded-buffer
(Fig. 3) Bounded-buffer

2) 프로세서 구성 및 할당

한 병렬 시스템에서 그 시스템을 구성하는 프로세서들(processors)이 다를 경우, 어떠한 프로세서에 어떤 객체를 할당하는가에 따라서 실행의 효율이 좌우될 수 있다. 이에 따라 VEPO에서는 사용자가 원하는 프로세서상에 객체를 할당할 수 있도록 시각적으로 표현하는 것을 지원하고 있다. CONFIG 메뉴의 CREAT 부분을 선택하여 창을 생성한 후, 프로세서 아이콘을

사용하여 시스템의 구성 형태와 각 프로세서상에 할당할 객체들을 표현한다. 또한 각 프로세서의 하드웨어 사양을 기술할 수 있으며, CONFIG의 hardware specification을 선택하면 이를 기술하기 위한 창이 생성된다. 만약, 위와 같이 프로세서상에 객체들을 할당해주는 부분을 명시적으로 표현하지 않는다면 시스템에서 자동적으로 임의의 프로세서에 객체들을 할당해준다.

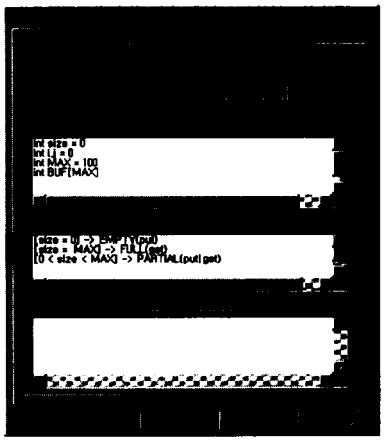
3) 프로그램 수행

작성된 프로그램을 수행하기 위해서 BUILD 메뉴를 선택한다. 한 프로그램은 BUILD 과정을 거치면서 먼저, POOSL로 변환된 후 Inmos C 코드로 변환된다. 변환된 Inmos C코드는 번역 과정을 통하여 목적 코드로 번역된다. 그런 다음 RUN을 선택하면 프로세서 구성 및 할당 그래프에서 지정된 사항에 맞게 각 프로세서에 할당되어 수행된다.

3.2.2 부가 정보 창

부가 정보 창은 그래픽 작업 창에서 구성된 객체 노드들의 수행에 필요한 정보들을 기술하기 위해 제공되는 창이다. 이러한 부가 정보 창은 객체의 인터페이스 부분의 정보를 기술하기 위한 것과 메소드의 구현 코드를 입력하기 위한 것이 있다.

각 창은, EDIT 메뉴의 Interface information과 Implementation information을 선택하거나 각 객체의 이름과 메소드 위에서 OPEN 아이콘을 클릭하는 경우 생성된다. (그림 4)는 객체의 인터페이스 부분을 기술하기 위한 부가 정보 창으로써, variable, method synchronization, data distribution 필드들을 포함하고 있다. Method synchronization 필드는 프로그램을 수행중 객체간의 통신이 발생될 때 동기화를 제어하기 위해, 메시지 요구를 받은 객체가 그 객체의 현재 상태에 따라 해당 메소드를 구분하여 수행할 수 있도록 동기화 규약을 표현하는 부분이다. Data distribution 필드는 앞에서 언급하였던 자료 병렬성에 관련된 것으로써 노드 모듈들에 자료를 분배하는 코드를 기술하는 부분이다. 여기서 기술되는 정보는 POOSL에서 정의된 고수준의 구문을 사용하여 작성된다. (그림 4)에서 버퍼 객체에 대한 부가 정보 창의 method synchronization 필드에는 동기화 코드가 기술되었으며, 이것은 버퍼가 다 차있을 경우에는 get 메소드만 수행 가능하고, 버퍼가 비어 있을 경우에는 put 메소드만이 수행 가능하다는 것을 의미한다. 이러한 동기화 정보는 실행 모델에서 객체의 통신과 동기화를 제어하는 객체 관리기(object manager)에 의해 동기화를 제어하는데 이용된다.



(그림 4) 부가 정보 창
(Fig. 4) Embedded Information Window

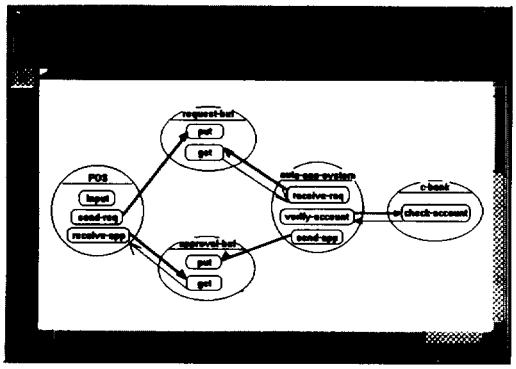
3.3 예제 프로그램

이 절에서는 예제 프로그램으로써 자동 결재 시스템(automatic approval system)과 행렬 곱(matrix multiplication) 문제를 설명한다. 이 두가지 예제 프로그램은 각각 다른 특징이 있다. 병렬 객체지향 언어들에 대한 연구에서 가장 대두가 되는 부분이 동시에 수행되는 객체들간의 동기화 문제이다[1,4,5]. 이러한 동기화 문제를 설명하기 위해 대부분의 언어들에서 보여주는 예제가 3.2절에서 설명한 bounded-buffer 문제이다. 이 절에서는 이 동기화 문제를 위해 위의 bounded-buffer 문제를 확장한 예로써 자동 결재 시스템을 설명하고 있다. 또한, 두번째 예로써 자료 병렬성을 보여주기 위한 예로서 행렬 곱 문제를 설명하고 있다. 행렬 곱 문제의 특징은 각 벡터에 대한 내적을 동시에 구할 수 있기 때문에, 각 프로세스들이 서로 다른 데이터를 가지고 동일 연산을 수행하는 자료 병렬성을 특징을 잘 반영한다.

3.3.1 자동 결재 시스템

자동 결재 시스템은 POS 단말기를 통해 고객이 상품을 구매하려고 할 경우 고객의 은행 계좌를 검사하여 구매 상품에 대한 지불 능력이 있는지를 검사하여

구매 승인을 내려주는 시스템이다. (그림 5)에서 볼 수 있는 바와 같이 POS(Point-of-Sale) 객체, request- buffer 객체, approval-buffer 객체, auto-app-system 객체, c-bank 객체들로 구성된다. 이 객체들 중 POS 객체와 auto-app-system 객체는 능동 객체로, request-buffer 객체, approval_buffer 객체, c-bank 객체는 수동 객체로 정의되었다.

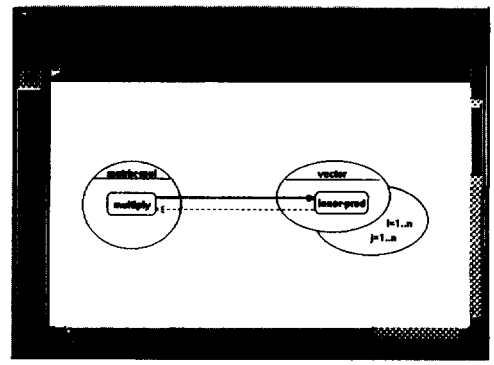


(그림 5) 자동 결재 시스템
(Fig. 5) Automatic Approval System

POS 객체는 거래량, 거래 물건명, 고객의 계좌 번호, 상점의 계좌 번호와 같은 상거래에 필요한 정보들을 입력 받는다. 그런 다음 request-buffer에 거래 요구를 전달하기 위해 request-buffer의 put 메소드를 호출한다. Request-buffer는 POS 객체로부터 메시지를 받고 put 메소드를 수행한다. 한편, auto-app-system 객체는 거래 요구를 받아 그 거래에 대한 고객 계좌를 검사하여 거래 승인을 전달하는 객체로써, 거래 요구를 받기 위해 request-buffer 객체의 get 메소드를 호출한다. Request-buffer에 거래 요구가 있을 경우, 거래 요구를 받아 그 고객의 계좌를 조사하기 위해 c-bank 객체의 check-account 메소드를 호출한다. c-bank 객체는 고객의 은행 객체로써 고객의 계좌가 거래 요구 상품에 대한 비용을 지불할 정도로 충분한지를 검사하여 정보를 제공한다. Auto-app-system 객체는 c-bank로부터 충분하다는 정보를 받은 경우에 거래 승인을 POS 객체에 보내기 위해 approval-buffer의 put 메소드를 호출한다. 한편, POS 객체는 approval-buffer로부터 거래 승인을 얻기 위해 get 메소드를 호출하게 된다. 이때, approval-buffer는 POS 객체로부터 메시지를 받은 후, 버퍼에 거래 승인이 있을 경우 get 메소드를 수행한다. 이 예제에서도 양 버퍼에 메시지가 요청되었을 경우, 버퍼가 다 차있을 경우와 비어있을 경우를 검사하여 객체간의 동기화를 제어하게 된다.

3.3.2 행렬 곱

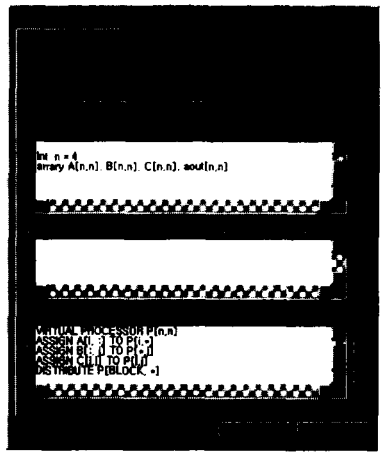
행렬 곱에 대한 프로그램은 두 개의 객체 matrix-mul과 vector로 정의되었다. Matrix-mul은 호스트 객체로써 연산을 수행하는데 필요한 데이터를 각 노드 객체들에 할당하며, 이를 위해 부가 정보 창의 data distribution 필드에 정보를 기술한다. 노드 객체인 vector 객체는 호스트 객체로부터 분배된 데이터를 가지고 각 벡터의 내적을 구하는 inner-prod 메소드가 정의되었다. (그림 6)에서 볼 수 있는 바와 같이 vector는 (1..n, 1..n)의 복제 객체 아이콘으로 정의되었으며, 이는 n * n개의 메쉬(mesh) 구조를 구성하는 가상 프로세서들이 동시에 vector 객체를 수행하는 것을 나타낸다. Matrix-mul의 메소드 multiply는 vector의 inter-product 메소드를 호출하고 있으며, 이는 각 가상 프로세서들이 inter-product 메소드들을 동시에 수행하도록 한다. 따라서 각 가상 프로세서들은 메소드 호출을 받고, 각기 할당된 데이터를 가지고 벡터의 내적을 구하게 된다. 각 vector 객체들은 각각의 벡터의 내적을 구한 후에, matrix-mul에 결과를 보낸다.



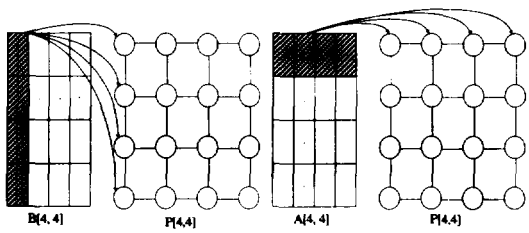
(그림 6) 행렬 곱
(Fig. 6) Matrix-multiplication

(그림 7)은 matrix-mul의 부가 정보 창으로써 데이터 분배를 위한 코드가 기술되었다. Data distribution 필드에서 먼저 가상 프로세서 P[n, n]가 정의되었고, 이 가상 프로세서에 데이터를 할당시키기 위한 것으로써, 행렬 A의 i행의 모든 요소(A[i,*])를 i행에 위치하

는 모든 가상 프로세서상(P[i,:])에 복사하며, 행렬 B의 j열의 모든 요소(B[*,j])를 j열에 위치하는 모든 가상 프로세서상(P[:,j])에 복사한다. 이에 대한 과정이 (그림 8)에 보여진다. 이 자료 병렬성에 관련된 명령어들에 대한 자세한 설명은 [3]에서 참조될 수 있다.



(그림 7) Matrix-mul의 부가 정보 창
(Fig. 7) Embedded Information Window for Matrix-mul



(그림 8) 가상 프로세서상에 데이터 분배
(Fig. 8) Data Distribution on Virtual Processors

3.4 VEPO 수행 모델

본 절에서는 VEPO의 수행 모델에 대해 설명한다. VEPO의 수행 모델에서는 동시에 수행되는 객체들의 통신과 동기화를 제어하기 위해서 객체마다 객체 관리기를 유지하고 있다. 이 객체 관리기는 번역과정 중에 자동적으로 생성되며, 실행 시간에 활성화된다.

3.4.1 객체 관리기

객체 관리기의 기능은 크게 객체간의 각 통신 방법들을 제어해주는 기능과 통신에 따른 동기화를 제어해주는 기능으로 구분될 수 있다.

VEPO에서는 동기적인 통신 방법뿐만 아니라, 비동기적인 통신 방법과 future variable을 이용한 미래형 방식의 통신까지 지원하기 때문에, 객체 관리기는 각 통신방법들을 실제적으로 제어해주고 있다. 또한, 동기화 문제와 관련해서 객체 관리기는 통신이 수행될 때 intra-object 동시성과 부가 정보 창에 기술된 method synchronization 정보를 토대로 객체의 현재 상태와 외부로부터 들어오는 모든 메시지들을 관리하면서 객체간의 동기화를 제어해주고 있다. 특히, 상속성 이상이 발생되지 않도록 동기화를 제어하고 있다[16].

1) 객체 관리기의 구성 요소

(그림 9)는 객체 관리기가 관리하는 각각의 구성 요소들에 대한 설명은 다음과 같다.

- 메시지 큐(Message queue)
  외부로부터 전송 받은 메시지에 대해 객체가 현재 수행할 수 없는 상태이거나, 요구된 메소드가 수행중인 경우에 메시지를 메시지 큐에 보관하게 된다.
- 메소드 상태 테이블(Method state variable)
  객체내의 각 메소드에 대한 상태 정보를 보관한다. 메소드의 상태 정보는 세가지 상태를 유지할 수 있으며, 현재 다른 메시지의 요구에 대해 수행 중인 경우에는 block 상태, 수행하지 않는 경우에는 free 상태로 나타내며, 가장 최근에 수행을 마친 메소드인 경우에는 m-free로 나타낸다.
- 객체 상태 테이블(Object state table)
  객체의 현재 상태와 그 상태에서 수행될 수 있는 메소드들, 각 메소드 집합에 속하는 메소드들이 수행되기 전에 만족되어야 할 상태 정보를 나타내는 가아드 조건식을 포함한다.
- 미래형 변수 박스(Future variable box)
  미래형 방식의 통신의 경우에 수신 객체로부터 결과값이 도착한 경우, 후에 참조할 수 있도록 결과값이 들어있는 메시지를 보관하기 위해 이용한다. 자료 구조는 객체에서 사용되는 future variable 이름과 future variable을 이용하는 메시지들의 리스트로 구성된다.
- 채널 정보 박스(Channel information box)
  객체 관리기와 객체의 메소드간의 통신을 위해 사용되는 소프트 채널(soft channel) 번호와 외부 객체와의 통신을 위한 하드 채널(hard channel) 번호

를 저장한다. 이 정보는 객체 관리기가 통신을 제어할 때 참조한다.

```
O : an object, M : a method, OST : object state table, MST :
method state table, MQ : message queue, r.m : a requested
message from other objects, s.m : a serviceable message in the
current object state, p.m : a previous message processed, c.o.s :
current object state, n.o.s : new object state

loop
  wait for a r.m from other Os or a signal from a completed M
    if there is a signal from a completed M
      #method-counter--;
      if #method-counter is zero
        convert the state of method that is block to m-free
        convert the state of method that is m-free to free
      if there is a change of c.o.s
        marks "T" into n.o.s
      while there are s.ms in MQ
          gets a s.m from MQ
          services the s.m
      end while
    else if there is a r.m
      get the r.m
      if the r.m is acceptable in c.o.s
        checks MST
        call the routine for intra-object synchronization
      else
        inserts the r.m in MQ
end loop
```

(그림 9) 객체 관리기의 수행 절차
(Fig. 9) Procedure of Object Manager

2) 객체 관리기의 수행 과정

한 객체가 다른 객체로 부터 메소드 호출을 요구받을 때, 객체 관리기는 먼저 그 메소드 호출이 현재의 객체 상태에서 수행될 수 있는지를 체크한다. 이를 위해 객체 상태 테이블에서 객체의 현재 상태와, 그 상태에서 실행 가능한 메소드 집합에 그 메소드 호출이 포함되는지를 체크한다. 만약, 객체가 그 메소드를 수행할 수 없는 경우라면, 그 메소드 호출은 나중에 실행되기 위해 메시지 큐에 저장된다. 그 객체가 메소드를 수행할 수 있는 상황이라면, 객체 관리기는 메소드 호출을 처리하기 위해 intra-object 동기화 프로시쥬어를 수행한다. 또한 이 동기화 프로시쥬어를 수행하기 위해서 method state table이 참조된다. 객체 내부의 메소드가 수행을 마치면 객체 관리기로 끝났음을 알리는 신호를 보낸다. 객체 관리기는 메소드로부터 수행이 끝났음을 알리는 신호를 받고, 활성 프로세스의 갯수를 감소시킨다. 만약, 한 메소드에 대한 활성 프로세스의 갯수가 0이 되면 그 메소드의 상태는 block상태에서 m-free상태로 바뀐다. 또한 메소드의 상태가 m-free인 것은 free 상태로 바뀐다. 그런 다음, 메시지 큐에 대기하는 메시지 중 새로운 객체의 상태에서 수행 가능한 메시지를 찾아서 수행한다. 이때, 동일 메소드에 대해 대기하는 메시지 간에는 FIFO(First-In First-Out) 방식으로 처리해 준다.

객체간의 통신은 각 객체의 채널 사이에 연결된 하드웨어 링크(link)를 이용하여 통신하며 객체내의 객체 관리기와 메소드 간에는 메모리를 이용한 소프트 채널을 이용하여 통신한다. 객체간의 통신에 있어서 객체에 들어오는 모든 메시지는 객체 관리기가 관리하게 된다. 객체 관리기는 항상 채널을 감시하고 있다가 어떤 채널로 메시지가 도착하면 그 메시지를 입력받게 되는데, 채널을 감시하는 것은 ProcAltList 함수를 이용하며 메시지를 입력 받는 것은 ChanIn 함수를 이용한다. ProAltList와 ChinIn 함수는 Inmos C에서 제공하는 함수로써, ProAltList는 프로세서의 입력 채널들 중 한 곳으로 데이터가 도착했을 경우, 그 해당 채널을 알려주는 유용한 함수이다. 객체 관리기에 대한 수행 절차는 (그림 9)와 같다.

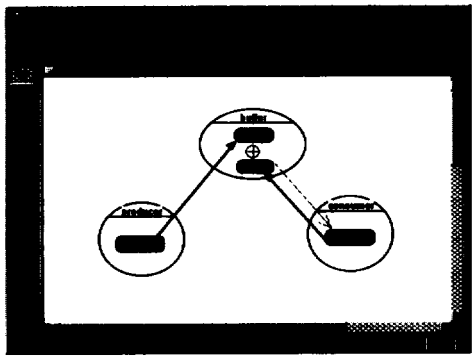
3.5 실행 과정의 시각화

VEPO는 프로그램의 실행 과정을 시각적으로 보여줄 수 있으며, 이 과정은 그래픽 작업 창에서 작성된 본래의 시각 프로그램을 이용하여 보여준다.

그래픽 작업 창의 프로그램 그래프을 살펴보면, 한 객체가 수행될 때 그 객체안에 정의된 메소드들의 색깔이 변한다. 수행전에는 흰색이었던 것이 수행중에는 녹색으로 바뀌고 수행 후에는 녹색에서 파란색으로 바뀌게 된다. 다른 메소드가 수행을 시작하게 되면 마찬가지로 그 메소드도 흰색에서 녹색으로 바뀌게 되며 수행 후에는 파란색으로 바뀌게 된다. 이때 그전에 수행을 마쳐 파랑색이었던 메소드가 흰색으로 바뀌게 된다. 이렇게 메소드 아이콘의 색상이 녹색에서 흰색으로 바뀌는 과정에, 중간에 파란색으로 거치는 이유는 그 메소드가 가장 최근에 수행되었다는 것을 나타내기 위함이다. 이것은 상속성 이상의 문제를 해결하는 것과 관련이 있다[16]. 메소드의 색깔이 빨간색으로 변하는 경우는 데드락(deadlock)과 같은 문제가 발생했을 때이다. (그림 10)은 3.2절에서 설명한 bounded-buffer 문제를 실행시킬 때의 과정을 보여주고 있다. 현재 buffer 객체의 get 메소드가 수행중으로 메소드의 색깔이 녹색이고, put 메소드는 수행된 상태로 파란색을 나

타내고 있다.

이러한 프로그램의 실행 과정에 관한 추적 정보는 파일로 저장되며, TRACE 메뉴를 선택하여 여러 형태로 수행시킬 수 있도록 현재 구현중에 있다. 그래픽 작업 창의 상단에 이와 관련된 아이콘들이 정의되어 있다. 이 아이콘들은 각각 맨처음으로 되돌아 갈 수 있고(rewind), 거꾸로 실행 시점을 되돌릴 수 있고(backword), 현재 시점에서 어떤 시점까지 앞으로 건너뛸 수 있고(forward), 실행을 중단시킬 수 있고(stop), 수행을 진행할 수 있다(play)는 것을 의미한다.



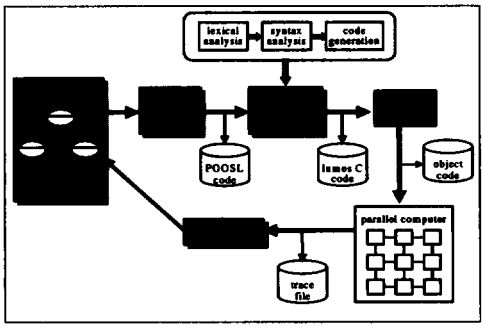
(그림 10) Bounded-buffer의 실행 과정
(Fig. 10) Execution Animation of Bounded-buffer

4. VEPO 시스템의 구현

이 장에서는 VEPO 시스템의 전반적인 수행 과정과 구현에 대해 설명한다.

4.1 VEPO 시스템의 수행 과정

(그림 11)은 VEPO 시스템의 전반적인 수행 과정을 보여준다. 사용자는 VEPO 그래픽 편집기에서 아이콘을 사용하여 시각 프로그램을 작성한다. 또한 부가 정보 창을 호출하여 계산식과 변수값 등 텍스트 정보를 입력한다. 이러한 텍스트 정보는 본 연구의 선행 연구로써 이미 개발된 POOSL로 작성된다. 이 VEPO 프로그램의 시각 객체들은 먼저, POOSL 코드로 번역된 후, Inmos C 소스 코드로 생성된다. 이 생성된 코드는 다시 컴파일 과정과 링크, 로드 과정을 거쳐 병렬 컴퓨터의 각 프로세서에 할당되어 실행된다. 이때 실행 과정은 VEPO 그래픽 편집기에서 작성되었던 원시 프로그램 상에 시각적으로 보여진다.



(그림 11) VEPO 시스템의 수행 과정
(Fig. 11) Process of VEPO System

4.2 구현 환경

VEPO 그래픽 사용자 인터페이스는 PC에서 Visual C++를 사용하여 구현되었으며, VEPO 시스템의 구현 환경은 PC를 호스트로 연결한 트랜스퓨터들로 구성된 병렬 컴퓨터 MC-3이다. MC-3는 18개의 RISC-type 마이크로 프로세서인 T800 트랜스퓨터로 구성된 병렬 시스템으로, 각 트랜스퓨터는 4MB의 지역 메모리를 가지고 있으며 통신을 위한 4개의 채널을 가지고 있다. Inmos C는 CSP에 기반을 둔 언어로, 트랜스퓨터를 기초로 한 병렬 시스템에서 실행되는 언어이다. 이 언어는 기존의 순차형 C언어에 라이브러리를 이용하여 병렬성을 추가한 언어로써, 병렬 프로그래밍을 위한 환경을 제공할 뿐만 아니라 기존의 순차형 C 언어의 강력한 기능들도 모두 포함하고 있다.

4.3 Inmos C 코드 생성

VEPO에서 작성된 시각 프로그램은 먼저 POOSL로 번역된 후, 어휘 분석(lexical analysis), 구문 분석(syntax analysys), 코드 생성(code generation)의 세 단계를 거쳐서 MC-3상에서 실행 가능한 Inmos C 코드로 변환된다. 어휘 분석과 구문 분석을 수행하면서 AST (Abstract Syntax Tree)를 생성되며, AST는 프로그램의 구성요소를 나타내는 노드와 이러한 노드를 연결하는 포인터들로 구성된다. 특히 코드 생성 단계에서는 병렬 프로그램이 수행되는데 필요한 통신과 동기화에 관련된 복잡한 코드들이 자동적으로 생성된다.

4.3.1 객체의 번역

한 객체는 Inmos C의 한 linked unit 즉 메인 프로그램으로 변환된다. 이 과정에서 생성되는 코드는 다

음과 같다.

- 채널을 생성하고 할당해 주는 코드
  외부 객체와의 통신을 위한 하드 채널과, 객체 내의 프로세스간의 통신을 위한 소프트 채널을 생성하고 할당해 주는 코드를 생성한다.
- 프로세스를 생성하고 할당해 주는 코드
  객체에서 프로세스로 할당되는 모듈은 각 메소드들과 번역기에 의해 생성되는 객체 관리기이다. 각 메소드와 객체관리기는 Inmos C의 함수 형태로 번역되며, 이 함수는 Inmos C의 한 프로세스로 할당되어 수행된다.
- 통신에 대한 코드
  통신은 Inmos C의 ChanIn, ChanOut 함수로 번역되며, 메시지 프레임 단위로 데이터를 주고 받는다.

4.3.2 객체 관리기의 생성

객체 관리기 함수는 라이브러리(library) 형태로 구현되었으며, 번역과정에서 객체 관리기가 관리하는 각종 테이블과, 테이블에 적절한 정보를 삽입해 주는 코드가 생성된다.

4.3.3 프로세서들 구성 및 프로세스 할당 코드의 생성

Inmos C코드로 번역된 프로세서의 구성 및 프로세스 할당에 관한 코드를 입력으로 받아, 시스템의 구성형태에 따른 각 프로세서의 정의, 프로세서간의 통신을 위한 트랜스퓨터 링크에 하드 채널의 할당 및 연결에 관한 코드를 생성해준다. 또한 각 프로세서에서 수행될 프로세스를 할당시키는 코드를 생성한다. 이때 시스템의 구성형태가 regular 구조인 경우와 irregular 구조인 경우에 따라 구현 방법이 달라진다[3]. Irregular 구조인 경우, 생성된 코드는 hardware description, software description, 그리고 mapping description으로 구성된다. Hardware description은 프로세서를 정의하고 프로세서간의 물리적 링크를 연결해 주는 부분이다. Regular 구조는 각 시스템의 종류들에 따라 라이브러리 형태로 구현되어 있어, 지정하는 시스템 형태에 따라 자동적으로 시스템을 구성해준다.

4.3 실험 및 결과 분석

본 논문에서 구현된 VEPO에 대한 결과 분석을 위해 프로그램의 길이 및 복잡도 면에서 고려하였다. 예제 프로그램은 bounded-buffer 문제에 대해서 VEPO와 다른 병렬 언어를 사용하여 수행시켜 본 후 비교하였다. 비교할 병렬 언어들은 본 시스템의 구현 환경인 병렬 컴퓨터 MC-3 상에서 수행가능한 병렬 언어인 Inmos C와 Occam이다.

VEPO에서는 3.2절의 (그림 3)과 (그림 4)에서 볼 수 있는 것처럼, bounded-buffer 문제가 간단하게 작성될 수 있지만, Inmos C와 Occam 언어인 경우에는 프로그램 라인 수가 각각 195, 151이었다. 또한 프로그램의 복잡도 면에서 살펴볼 때, Inmos C와 Occam 언어로 작성된 프로그램을 분석해보면, 병렬 처리를 위한 프로세스들의 생성, 프로세스간의 통신을 위한 채널의 할당과 연결, 프로세스간의 동기화를 제어하기 위한 코드들이 포함되기 때문에 프로그램이 길어지고, 복잡해진다. 그렇지만, VEPO에서는 이러한 코드들이 번역중에 자동적으로 생성되기 때문에 프로그램 작성이 간편해지고, 작성된 프로그램을 이해하는 것도 쉽다는 것을 알 수 있다.

5. 결 론

본 연구는 사용자가 병렬 프로그램을 쉽게 작성하고 수행시킬 수 있는 병렬 시각 프로그래밍 환경 VEPO를 제안하였다. 본 연구와 기존의 병렬 시각 환경들 VPE, CODE, HENCE과의 차이점은 먼저 기본 프로그래밍 모델이 다르다. 본 연구에서 제안하고 있는 VEPO는 객체지향 모델에 기초하고 있기 때문에 병렬성과 관련되는 객체지향의 특징들을 지원하고 있다. VEPO에서 프로그램의 한 수행 단위를 객체 노드로써, 수행 단위간의 통신은 메시지 전달 방법으로 나타낸다. 이와 같이 VEPO는 객체지향 개념 위에서 시각 프로그램을 작성하기 때문에 주어진 문제에 내재된 병렬성을 좀더 자연스럽고 쉽게 표현 할 수 있으며, 고성능 계산에 적합한 자료 병렬성을 지원하고 있기 때문에 효과적인 병렬 처리를 수행할 수 있다. 또한 작성된 본래의 시각 프로그램을 이용하여 프로그램의 실행 과정을 시각적으로 보여줌으로써 데드락과 같은 에러를 추적할 수 있다.

무엇보다도, 본 VEPO 시스템에서는 병렬 프로그램에서 복잡하고 오류가 발생하기 쉬운 프로세스간의 통신과 동기화에 관한 코드를 사용자가 직접 작성할 필요없이 번역 과정에서 자동적으로 생성하도록 하고 있다. 따라서, 사용자는 병렬 프로그램의 복잡하고 까다

로운 코드들을 직접 작성해야 하는 부담감으로부터 벗어나 프로그램의 구성과 설계에 더 많은 관심을 갖게 됨으로써 좀더 잘 계획된 프로그램을 작성할 수 있다는 장점이 있다. 현재까지 병렬 객체지향 개념을 기초로 한 시각 환경에 대한 연구는 국·내외적으로 없는 상태이다.

그러나, 현재 본 시스템의 수행 과정을 살펴보면, VEPO에서 작성된 시각 프로그램은 POOSL 코드로 번역되는 과정을 거쳐 Inmos C 코드가 생성되기 때문에, 실행의 효율이 떨어질 수 있다. 따라서, 앞으로 POOSL 코드로 번역되는 과정을 거치지 않고 직접 시각 프로그램으로부터 Inmos C 코드로 번역될 수 있도록 구현되어야 한다. 또한, 프로그램의 성능 분석을 지원하고, 그 성능 분석의 결과에 따라서 효과적으로 병렬 컴퓨터 상에 프로세스들을 할당해 주는 방법들에 대한 연구도 요구된다.

참 고 문 헌

[1] S .Y. Steplen, et al., "PROOF : A Parallel Object-Oriented Functional Computational Model," Journal & Distributed Computing, Vol.12, No.3, pp.202-212, 1992.

[2] 최숙영, 권혁찬, 유관종, "트랜스퓨터를 기초로 한 시스템에서 효율적인 병렬처리를 위한 객체지향 프로그래밍 모델", 한국 정보과학회 논문지(C), 제2권 제2호, pp.115-129, 1996.

[3] 최숙영, '병렬 객체지향 표기 언어 POOSL의 설계', 박사학위 논문, 1996.

[4] A. Tripathi and E. Berge, "An Implementation of the Object-Oriented Concurrent Programming Language SINA," Software-Practice and Experience, Vol.19, No.1, pp.235-256, 1989.

[5] A. Yonezawa, et al., "Object-Oriented Concurrent Programming in ABCL/1," in Proc of OOPSLA, pp.258-268, Sept., 1986.

[6] J. C. Brown, et al., "Visual Programming and Debugging for Parallel Computing," IEEE Parallel & Pistributed Technology, pp.75-83. 1995.

[7] G. Wirtz, "A Visual Toolset for Messag Passing Parallel Programming," in Proc. of PDPTA Conference, pp.301-311, 1985.

[8] 김상욱, 박지은, "시각적 사용자 인터페이스 언어의 개발", 한국 정보과학회 논문지, 제20권, 제11호, pp.1632-1648, 1993.

[9] E. P .Glinert, "Pict : An Interactive Graphical Programming Environment," IEEE Computer, Vol.17, No.11, pp.7-25, 1984.

[10] M. Hiraeawa, et al., "Iconic Programming System HI-VISUL," IEEE Trans. on Software Engineering, Vol.16, No.10, pp.1178-1184, 1990.

[11] P. Newton and J. Dongrarra, "Overview of VPE : A Visual Environment for Message-Passing Parallel Programming," Tech. Report UT-CS-94-261, Computer Science Dept., Univ. of Tennessee, Knoxville, Tenn., 1994.

[12] P. Newton and J. C. Browne, "The CODE 2.0 Graphical Parallel Programming Language," in Proc. of ACM Supercomputing Int. Conference, pp.1024-1030, July, 1992.

[13] P. Newton, "A Graphical Retargetable Parallel Programming Environment and Its Efficient Implementation," Tech. Report TR93-28, Dept. of Computer Sciences, Univ. of Texas, Austin, Tex., 1993.

[14] A. Beguelin, et al., 'The HeNCE : A User's Guide Version 2.0', 1994.

[15] A. Beguelin, et al., "Visualization and Debugging in a Heterogeneous Environment," Computer, Vol. 26, No.6, pp.88-95, 1993.

[16] S.Y. Choi, et al., "A Synchronization Mechanism for Inheritance in POOSL," in Proc. of High Performance Computing Asia Int. Conference, pp.708-711, May, 1997.

최 숙 영

e-mail : sychoi@core.woosuk.ac.kr

1988년 8월 전북대학교 이학사(전산학)

1991년 2월 전북대하교 이학석사(전산학)

1996년 2월 충남대학교 이학박사(전산학)

1996~현재 우석대학교 조교수

관심분야 : 멀티미디어 시스템, 병렬 처리, 원격 교육, CAI