ParaC 언어의 설계 및 구현

이 경 석† · 우 영 춘† · 김 진 미† · 지 동 해†

요 약

본 논문은 공유 및 분산 메모리 구조를 가진 병렬 컴퓨터의 프로그래밍 환경을 지원하기 위하여 ParaC 언어를 설계하고 구현한 내용을 기술한다. ParaC 언어는 확장성 높은 병렬 컴퓨터의 시스템 자원을 사용자가 효과적으로 이용할 수 있도록 설계되었다. 이것은 C 언어에 공유 메모리 환경을 위한 병렬 구문과 동기화 구문, 그리고 분산 메모리 환경을 위한 원격 태스크 구문을 추가함으로써 이루어졌다. 언어의 구현을 위하여 C 언어로의 번역 방법을 기술하였으며, 이 방법을 사용한 번역기와 확장 구문을 위한 실행시간 라이브러리를 구현하였다.

The Design and Implementation of the ParaC Language

Kyoung Seok Lee† · Young Choon Woo† · Jin Mee Kim† · Dong Hae Chi†

ABSTRACT

This paper describes the design and implementation of the ParaC language that supports parallel programming on the shared memory and distributed memory parallel machine. The ParaC language is designed for the effective use of system resources of scalable parallel systems. The goal is achieved by adding parallel and synchronization constructs for shared address spaces, and remote task constructs for distributed address spaces. This paper also shows the translation method, and we implement the translator and the run-time library for parallel execution of extended constructs.

1. 서 론

병렬 처리 기술의 발달로, 과거에 과학계산 분야 같은 특수 목적 분야에서 주로 사용되어 왔던 병렬 시스템은 확장성, 안정성, 수행 성능이 뛰어난 장점을 기반으로 응용 분야가 점차 범용으로 확장되고 있다. 그러나, 병렬 처리 기술의 발달에 비해 병렬 시스템을 사용하기 위한 병렬 프로그래밍 환경에 대한 연구는 미약하여 일반 사용자의 병렬 시스템 사용을 어렵게 하고 있다. 특히, 병렬 환경은 시스템의 구조에 맞는 병렬성 표현 및 동기화 처리가 필요하기 때문에 순차 환경에 익숙한 사용자에게 프로그래밍의 어려움을 주고 있다. 그러므로, 병렬 시스템을 효과적으로 사용할 수 있도록 하기 위해서는 사용자에게 적합한 프로그래밍 환경을 제공하여야 한다.

병렬 프로그래밍 환경을 지원하기 위해 이루어지고 있는 연구 방법으로는 기존의 순차 언어로 작성된 프로그램을 자동으로 병렬화하는 방법(automatic parallelization), 기존의 순차 언어에 병렬 기능을 추가하여 병렬성 표현을 지원하는 방법(parallel extension), 그리고 새로운 병렬 언어를 개발하는 방법이 있다.

그 중에서 기존의 순차 언어에 병렬 기능을 추가하

※본 연구는 1996년 정보통신부와 과학기술처에서 시행중인 "고속병렬컴퓨터(주전산기Ⅳ) 공동연구개발" 사업의 일 부분으로 연구되었습니다.

† 정 회 원: 한국전자통신연구원 컴퓨터연구단

논문접수: 1997년 3월 21일, 심사완료: 1997년 10월 6일

여 병렬 프로그래밍을 지원하는 방법은 기존의 순차 언어에 익숙한 사용자에게 쉬운 프로그래밍 환경을 제공하고, 사용자가 시스템의 구조에 적합한 병렬성을 명시적으로 표현하도록 지원하여 시스템의 병렬 기능을 효과적으로 이용하게 하는 방법으로 알려져 있다[1].

SPAX(Scalable Parallel Architecture based on X-bar network) 시스템은 범용 및 대용량 자료 처리용으로 개발된 병렬 컴퓨터이다[2]. 이 시스템은 확장성과 안정성을 고려하여 설계한 계층적 분산 구조를 가진 대규모 병렬 처리(MPP:Massively Parallel Processing) 시스템이며, 마이크로커널 기반의 UNIX 환경을 지원하는 운영체제가 탑재되는 구조로 되어 있다.

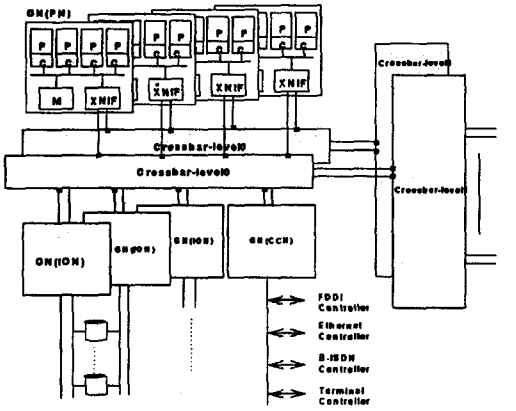
본 논문은 공유 및 분산 메모리 구조를 가진 SPAX 시스템의 병렬 프로그래밍 환경을 효과적으로 지원하기 위하여 ParaC 언어를 설계하고 구현한 내용에 대하여 기술한다.

ParaC 언어는 C 언어에 병렬 구문을 추가하여 공유 및 분산 메모리 환경의 시스템에서 병렬 프로그래밍을 할 수 있도록 설계된 언어이다. ParaC 언어의 설계는 병렬 시스템에 적합한 프로그래밍 모델을 정의하고 기존의 C 문법과 호환되는 구조의 병렬 구문을 추가하는 방법으로 이루어졌다. 또한, ParaC 언어의 구현은 공유 및 분산 메모리 환경의 시스템에서 언어를 구현하기 위해 고려하여야 할 내용을 정의하고, 이를 해결하기 위한 의미 동작(semantic action)을 설계한 후 ParaC 번역기를 구현함으로써 이루어졌다. 분산 메모리 환경을 위해서는, 시스템의 구조에 적합한 실행시간 라이브러리(run-time library)를 구현하여 실행 코드가 분산 환경에서 실행될 때 동작하도록 하였다.

본 논문은 먼저 2장에서 ParaC 언어의 목표 시스템인 SPAX 시스템의 구조와 운영체제를 간략하게 소개한다. 3장에서는 ParaC 언어를 설계한 내용에 대하여 기술하고, 4장에서 ParaC 번역 시스템을 구현하기 위한 방법 및 구현 내용에 대하여 설명한다. 5장에서는 구현된 번역 시스템의 시험과 본 논문의 연구와 관련된 연구 내용들을 비교 검토하였으며, 6장에서 결론을 기술하였다.

2. SPAX 시스템

SPAX 시스템은 확장성 높은 병렬 시스템의 개발을 목표로 범용 및 대용량 자료 처리용으로 개발된 컴퓨터이다. 이 시스템은 (그림 1)과 같이 Intel사의 Pentium pro 프로세서를 최대 4개까지 장착할 수 있는 GN(General Node)이 계층적 크로스바 스위치 연결망(crossbar switch network)으로 연결된 구조로 되어 있다. GN의 내부는 SMP(Symmetric MultiProcessor) 구조로 되어 있으며, 세부 구성에 따라 4개의 프로세서를 가지고 계산을 담당하는 PN(Processing Node), 입출력 장치와 통신 장치를 관리하는 ION(I/O Node)과 CCN(CommuniCation Node)으로 동작한다. GN은 네트워크상에서 NORMA(NO Remote Memory Access) 구조로 연결되어 클러스터를 구성하며, 클러스터는 16개까지 확장할 수 있도록 설계되었다.



(그림 1) SPAX 시스템 구조도
(Fig. 1) The SPAX architecture

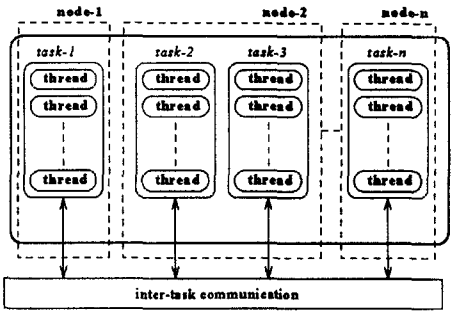
운영체제는 Chorus 마이크로커널(microkernel)에 UnixWare를 서버화한 구조로 된 UNIX이다. 시스템의 하드웨어 의존적인 부분은 마이크로커널이 담당하며, 서버에서 시스템 전체의 환경을 단일 시스템 이미지(Single System Image)로 관리하여 준다. 이것은 분산 메모리 구조인 노드간과 클러스터간에도 표준 시스템 호출의 지원, 단일 파일 시스템의 지원 같은 표준 UNIX 운영체제의 기본 기능을 지원하는 것을 의미한다.

3. 언어의 설계

본 장에서는 ParaC 언어의 설계 목표를 기술하고 언어를 설계하기 위한 프로그래밍 모델을 소개한다. 그리고 ParaC 언어를 정의한 내용을 각 구문별로 나누어 기술한다.

3.1 설계 목표

병렬 확장 언어인 ParaC 언어는 다음과 같은 요구조건을 가지고 설계되었다.



(그림 2) ParaC 프로그래밍 모델
(Fig. 2) The ParaC programming model

첫째, 확장성 높은 병렬 시스템에 적합한 프로그래밍 모델을 제공하는 언어의 설계이다. 병렬 시스템의 가장 큰 장점은 확장성이다. 이러한 확장성을 효과적으로 이용하기 위해서는 시스템의 구조에 적합한 프로그래밍 모델을 정의하고, 병렬화의 단위를 제한하지 않는 언어의 지원이 필요하다[3].

둘째, 기존의 ANSI C 언어와 호환되는 문법을 제공하는 것이다. 이것은 병렬 확장 언어로 구현된 프로그램의 호환성에 큰 영향을 미친다. 또한 C 언어의 장점인 재호출(recursion)이나 중첩(nesting) 표현을 제공함으로써 병렬성을 확장성 높게 표현할 수 있도록 하는 장점을 제공한다[4].

셋째, 일반 사용자에게 접근하기 쉬운 구문을 제공하는 언어의 설계이다. 병렬 프로그래밍 시의 문제점은 순차 프로그래밍에서는 사용하지 않는 병렬성 표현 및 동기화 처리의 표현이 프로그램의 안정성에 큰 영향을 준다는 것이다. 그러므로 병렬성 표현 및 동기화 처리를 위한 구문을 사용자에게 쉬운 형태로 제공하는 것이 필요하다.

3.2 프로그래밍 모델

ParaC 언어를 위한 프로그래밍 모델은 (그림 2)와

```
parallel-statement:   parallel { statement-list_opt }
pfor-statement:       pfor ( pfor-init-statement ; expression_opt ; expression_opt ) { statement }
pfor-init-statement:  declaration-specifiers init-declarator-list
spawn-statement:      spawn postfix-expression ( argument-expression-list_opt )
atomic-statement:     atomic statement
                      atomic ( primary-expression_opt ) statement
type-specifier:       lock
                      task
type-qualifier:       single
tcreate-statement:    tcreate-expression ;
                      unary-expression assignment-operator tcreate-expression ;
tcreate-expression:   tcreate ( string-constant )
tcall-statement:      unary-expression assignment-operator tcall-expression
tcall-expression:     tcall ( postfix-expression, identifier ( argument-expression-list_opt ));
tsend-statement:      tsend ( identifier, argument-expression-list_opt );
treceive-statement:   treceive ( identifier , argument-expression-list_opt );
```

(그림 3) ParaC 문법
(Fig. 3) ParaC syntax

같으며, 확장성 높은 병렬 시스템에 적합한 프로그래밍 환경을 제공하기 위해 공유 메모리 모델과 분산 메모리 모델을 결합한 형태로 구성되어 있다.

본 모델의 병렬 실행 단위는 태스크와 쓰레드로 나누어진다. 태스크는 독립 주소 공간을 가진 실행단위이며 하드웨어 시스템의 구조와 관계없이 실행 될 수 있다. 쓰레드는 태스크 내의 공유 메모리 환경에서 쓰레드간에 동기화 처리를 이루며 수행된다. 태스크는 분산 메모리 환경에서 다른 태스크와 메세지 전송(message passing) 형태의 자료 전송을 이루며 수행된다. 병렬화 단위(granularity)의 관점에서는 쓰레드는 fine grain과 medium grain 크기의 병렬 실행을 나타내고, 태스크는 coarse grain 크기의 병렬 실행을 나타낸다.

3.3 구문의 정의

ParaC 언어는 공유 메모리 환경의 병렬 프로그래밍을 위한 병렬 구문과 동기화 구문, 그리고 분산 메모리 환경의 병렬 프로그래밍을 위한 원격 태스크 구문으로 구성된다. 각 구문의 문법은 (그림 3)과 같이 ANSI C 문법에 추가된 형태이며 각 구문의 의미는 다음과 같다.

3.3.1 병렬 구문

공유 메모리 환경에서 fine grain 또는 medium grain의 병렬성을 표현할 수 있도록 설계한 구문이다. 이 구문은 병렬 구문간의 재호출 표현과 중첩 표현을 지원하여 병렬 표현을 확장성 있게 할 수 있도록 하며, 병렬 구문내에서 사용된 모든 변수의 의미가 ANSI C 문법을 따르도록 설계하여 C 언어의 특징인 포인터 변수의 사용과 재호출 표현을 모두 지원하도록 하였다. 병렬 구문의 실행 단위는 쓰레드로 정의하였으며 각 구문의 의미는 다음과 같다.

- parallel : parallel문 내에서 블록으로 지정한 실행문이 쓰레드로 할당되어 병렬로 수행된다. 병렬 수행된 각 쓰레드는 동시에 종료한다.
- pfor : pfor문 내에서 블록으로 지정한 실행문이 반복 횟수개의 쓰레드로 할당되어 병렬로 수행된다. 쓰레드의 생성 갯수는 실행시에 지정되며 병렬 수행된 각 쓰레드는 동시에 종료한다. 여기에서 반복 횟수를 표시하는 변수는 프로그램의 안정성을 위해 실행시에 변경할 수 없는 특성을 가진다.
- spawn : spawn문에 의해 지정된 함수 호출이 쓰레드로 할당되어 수행된다. 생성된 쓰레드의 종료는 제어하지 않으며 호출된 함수의 복귀값도 관리하지 않는다.

(그림 4)는 병렬 구문을 사용한 예제이다. 이 프로그램은 parallel 구문에 의해 C1, C2, C3 문장이 병렬 실행되며, C2와 C3은 다시 parallel 구문과 pfor 구문에 의해 C21과 C22,그리고 C31이 각각 병렬 실행된다. 또한 C4 문장의 func 함수는 spawn 구문에 의해 쓰레드로 할당되어 수행된다.

```
int data[100];
func (int count)
{
  int t;
  for (t=0; t < count;t++)
  printf ("%d * %d = %d\n", t, t, data[t]);
}
main (void)
{
  int count;
  parallel {
    data[0] = 0 * 0;                          /* C1  */
    parallel {                                /* C2  */
      data[1] = 1 * 1;                        /* C21 */
      data[2] = 2 * 2;                        /* C22 */
    }
    pfor (count=3; count<100; count++) {      /* C3  */
      data[count] = count * count;            /* C31 */
    }
  }
  spawn func (count);                         /* C4 */
}
```

(그림 4) 병렬 구문의 사용 예
(Fig. 4) An example of parallel constructs

3.3.2 동기화 구문

병렬 구문으로 작성된 프로그램이 병렬로 실행될 때 발생하는 동기화 문제를 명시적으로 처리할 수 있도록 설계한 구문이다. 이 구문은 사용자에게 쉬운 동기화 프로그래밍 방법을 제공하기 위하여 설계되었으며, 구조적인 구문만을 추가하여 컴파일 시에 문법 오류를 검사할 수 있도록 하였다. 동기화 구문은 병렬로 실행되는 쓰레드간의 동기화가 이루어지도록 하기 위해 쓰레드의 동기화 프리미티브를 사용하여 번역되도록 하였으며, 각 구문의 의미는 다음과 같다.

- atomic: atomic문에 의해 지정된 실행문이 병렬로 실행될 때, atomic문에 의해 지정된 실행문간에 배타적 실행이 이루어진다.
- lock: 변수 선언문 형태로 설계된 구문으로, atomic 문을 사용할 때 사용자가 동기화 변수를 지정할 수 있도록 한다. atomic문에 lock 변수를 지정하면, atomic문의 동기화 범위는 동일한 lock 변수를 사용한 부분으로 제한된다.
- single: 변수 선언문 형태로 설계된 구문으로, *type-qualifier*의 문법에 따라 변수를 단일 할당(single-assignment)의 의미를 가진 변수로 선언하는 구문이다. 단일할당 변수란 다음의 의미를 가지고 동작하는 것을 말한다.
  - 값이 할당되기 전의 참조: 값이 할당될 때까지 대기
  - 값이 할당되기 전의 값 할당: 값을 할당함
  - 값이 할당된 후의 참조: 일반변수와 같이 참조
  - 값이 할당된 후의 값 할당: 값을 할당할 수 없으며, 오류를 표시하고 종료

  이 구문은 변수를 사용하는 것만으로 프로그램의 동기화 처리가 이루어지도록 하여 주며, 병렬 구문 내에서 실행되는 함수의 매개 변수로 사용하였을 때는 결정적인(deterministic) 프로그램을 작성할 수 있도록 한다[5].

(그림 5)의 예제 프로그램은 parallel 구문에 의해 병렬로 실행되는 C1과 C2 문장이 lk 변수에 의해 상호배제가 이루어지는 것을 나타낸다. 또한, single 변수인 flag에 의해 C3 문장이 실행된 후에 C4가 실행되는 것을 나타낸다.

```
main (void) {
  lock lk;
  single int flag;
  int result, count = 0;
  parallel{
    atomic (lk) count++;        /* C1 */
    atomic (lk) count++;        /* C2 */
    flag = 1;                   /* C3 */
    result = flag;              /* C4 */
  }
}
```

(그림 5) 동기화 구문의 사용 예
(Fig. 5) An example of synchronization constructs

3.3.3 원격 태스크 구문

분산 메모리 환경에서 coarse grain 단위의 병렬성을 표현할 수 있도록 설계된 구문이다. 이 구문은 메세지 전송 모델을 기반으로 설계하여 실행 단위간의 독립성을 최대한 보장하도록 하였다. 또한 원격 태스크 구문은 병렬 구문과 결합하여 사용할 수 있도록 설계하여 확장성 높은 병렬 프로그램을 작성할 수 있도록 하였다. 병렬 실행의 단위는 독립 주소 공간을 가진 태스크이며, 태스크의 생성, 호출, 자료 전송을 위한 구문을 제공하도록 설계하였다. 실행 단위인 태스크는 구현될 시스템의 구조에 따라 프로세스 단위나 메세지 전송 구조의 태스크 단위로 구현될 수 있으며, 각 구문의 의미는 다음과 같다.

- task: 함수를 태스크 형으로 선언하도록 설계된 구문이며, 태스크 형으로 선언된 함수를 포함하는 파일은 실행시에 부태스크로 동작한다.
- tcreate: task 구문에 의해 선언된 부태스크가 실행될 수 있는 환경을 설정하도록 지정하는 구문이다. tcreate 구문에서는 부태스크의 위치를 지정하여 주며, 주태스크는 명시된 부태스크와의 통신을 위하여 포트를 설정한다.
- tcall: 태스크 형으로 선언된 함수를 호출하여 독립적으로 실행되도록 지정하는 구문이다. 지정된 함수는 부태스크로 동작하여 주태스크와 병렬로 실행된다. 부태스크의 명시는 tcreate 구문에서 전달된 task 식별자를 사용한다.
- tsend: 지정된 다른 태스크로 자료를 전송하도록 지정하는 구문이다. 자료 전송의 지정은 매개 변수 형태로 이루어지며 실제적인 전송은 포트를 통하여 이루어진다.
- treceive: tsend에 의해 전송된 자료를 수신하도록 지정하는 구문이다. 자료 수신의 지정은 매개 변수 형태로 이루어지며, 지정된 자료가 수신될 때까지 대기한다.

(그림 6)은 원격 태스크 구문을 사용한 프로그램의 예제이다. 프로그램은 3개의 파일로 이루어져 있으며, 주태스크 프로그램이 C6, C8의 부태스크를 tcreate 구문으로 C1, C2와 같이 지정한 후 tcall 구문으로 C4, C5 형태로 함수를 호출하여 부태스크에서 함수

를 실행하도록 한다. 호출된 함수는 부태스크에서 실행되며, tsend와 treceive 구문을 C7, C9와 같이 사용하여 자료를 전송한다.

```
/* 주태스크 프로그램: main.pc */
main (int argc, char *argv[])
{
   task pid;
   task cid;
   int n=1;

   pid = tcreate("/home/parac/producer");/* C1 */
   cid = tcreate("/home/parac/consumer");/* C2 */
   parallel {                            /* C3 */
      tcall(pid, producer(cid, n));      /* C4 */
      tcall(cid, consumer(pid, n));      /* C5 */
   }
}

/* 부태스크 프로그램: producer.pc */
task producer(task cid, int n)           /* C6 */
{
   int i;
   static int send_data=100;
   for(i = 0; i < n; i++)
      tsend(cid, send_data);             /* C7 */
}

/* 부태스크 프로그램: consumer.pc */
task consumer(task pid, int n)           /* C8 */
{
   int i;
   static int receive_data;
   for(i = 0; i < n; i++)
      treceive(pid, receive_data);       /* C9 */
}
```

(그림 6) 원격 태스크 구문의 사용 예
(Fig. 6) An example of remote task constructs

4. 언어의 구현

본 장에서는 3장에서 기술한 ParaC 언어를 구현하기 위해 고려하여야 할 내용과 이를 해결하기 위한 방법, 그리고 구현 내용을 각 구문별로 나누어 기술한다.

4.1 구현 방법

본 논문에서는 ParaC 언어로 작성된 프로그램을 병렬 실행을 위한 코드가 추가된 C 언어로 번역하는 방법을 사용한다. 번역은 문법 규칙(syntactic rule)에 의미 동작을 추가하여, 상향식(bottom-up) 구문 분석을 하는 도중에 의미 동작을 실행하여 단일 패스(one pass)에 번역 작업을 완료하는 구문 지시적(syntax-directed) 번역 방법을 사용한다[6].

병렬 실행을 위해 사용된 실행시간 라이브러리는 쓰레드와 태스크로 구분된다. 쓰레드의 구현을 위해 사용된 쓰레드는 POSIX 1003.4a에서 제안한 pthread 라이브러리이며, pthread에서 사용한 주요 함수로는 쓰레드 생성과 조인 등에 필요한 프리미티브이고, 동기화를 위한 자료 구조는 상호배제와 조건 변수이다[7]. 태스크의 구현을 위해서는 분산 메모리 환경을 위한 실행시간 라이브러리를 구현하였으며, 내부적으로는 SPAX 시스템에서 지원하는 통신용 시스템 호출을 기반으로 구현하였다.

ParaC로 작성된 프로그램을 병렬 실행을 위한 코드로 번역하기 위하여 고려하여야 할 주요 내용과 번역을 위해 적용한 방법은 다음과 같다.

4.1.1 병렬 구문

병렬 구문의 번역은 크게 두가지를 고려하여 구현하였다. 첫째, 병렬 구문에 의해 지정된 실행문이 쓰레드로 할당되어 병렬 실행되도록 한다. 또한 병렬 구문내에서 사용된 모든 변수가 C의 범위 규칙(scoping rule)을 따르도록 번역한다. 둘째, 병렬성을 계층적으로 표현할 수 있도록 중첩 표현된 병렬 구문이 순서에 맞게 병렬 실행되도록 번역한다.

이것을 위하여 다음과 같은 방법을 사용하였다.

첫째, 병렬 구문에 의해 지정된 실행문이 쓰레드로 할당되도록 하기 위해서, 병렬 구문에 의해 지정된 각 실행문을 함수로 변환하고, 변환된 함수가 쓰레드 생성 프리미티브인 다음의 함수에 의해 호출되도록 번역한다.

int pthread_create (pthread_t * thread, pthread_attr_t * attr, pthread_fun_t func, any_t arg);

병렬 구문내에 사용된 변수는 파싱시에 전역 변수와 지역 변수로 구분하고, 지역 변수의 범위 규칙이 쓰레드내에서 유지되도록 struct 타입으로 구조화한 후 함수의 매개 변수로 전달되도록 번역한다.

둘째, 중첩된 병렬 구문의 번역시에 쓰레드 함수의 할당 형태가 결정되는 시점은 병렬 구문을 트리 형태로 표현하였을 때, 최하위 노드 병렬 구문의 파싱이 완료된 시점이다. 그러나 쓰레드에 할당할 모든 병렬 구문의 정보는 최상위 노드 병렬 구문의 파싱이 완료

된 시점에서 결정되므로, 모든 병렬 구문의 파싱이 완료된 후에 실행 순서에 따라 쓰레드 생성을 위한 코드를 생성하도록 하였다.

pfor 문에서 동적으로 생성되는 쓰레드의 처리를 위해서는 C의 라이브러리 함수인 malloc을 사용하여 동적 저장 공간을 지정한 후 생성될 쓰레드의 포인터와 매개 변수를 저장하여 쓰레드의 생성과 소멸이 동적으로 이루어지도록 하였다. spawn 문의 번역은 지정된 함수를 하나의 쓰레드에 의해 호출되도록 코드를 생성하고 매개변수로 사용된 변수를 구조화하여 전달하도록 하였다.

4.1.2 동기화 구문

lock 구문과 atomic 구문의 번역은 구문이 사용된 문장을 pthread 프리미티브인 다음의 함수를 사용하여 직접적으로 변환하여 처리한다.

```
int pthread_mutex_lock (pthread_mutex_t *__mutex);
int pthread_mutex_unlock (pthread_mutex_t *__mutex);
```

single 구문은 변수 선언문 형태로 지원되므로 single로 선언된 변수가 쓰레드 내에서 사용될 때 동기화 동작을 하도록 변환하여야 한다. 또한 single 구문은 포인터 형의 변수 선언을 허용하기 때문에 이를 의미에 따라 정확하게 변환하여야 한다.

```
void _single_write (_CONTROL * ctr) {
  pthread_mutex_lock (&ctr->mutex);
  if (!((ctr->flag) & 0x01)) {
   ctr->flag |= 0x01;
   pthread_cond_signal (&ctr->cond);
  } else {
    printf ("ParaC: ERROR : Multiple Assignments to
    single\n");
    exit (-1);
  }
  pthread_mutex_unlock (&ctr->mutex);
}
void _single_read (_CONTROL * ctr) {
  pthread_mutex_lock (&ctr->mutex);
  if (!((ctr->flag) & 0x01))
    pthread_cond_wait(&ctr->cond, &ctr->mutex);
  pthread_cond_signal (&ctr->cond);
  pthread_mutex_unlock (&ctr->mutex);
}
```

(그림 7) 단일할당 구문을 위한 함수
(Fig. 7) Functions for the single construct

본 논문에서는 실행문내의 single 변수를 쓰레드의 동기화 동작을 하는 함수 호출 코드로 변환하는 방법을 사용하였다. 즉, 실행문의 파싱시에 single 변수를 사용 형태에 따라 읽기/쓰기로 나누고, (그림 7)의 함수를 기반으로 하는 함수 호출 형태로 변환하도록 하였다. 호출에 사용되는 함수는 변수의 타입에 따라 생성되며, (그림 5)의 C3, C4 코드는 다음의 함수로 변환된다.

```
int * _single_write_int (_single_int * value) {
   _single_write (&value → ctr);
   return &value → value;
}
int * _single_read_int (_single_int * value) {
   _single_read (&value → ctr);
   return &value → value;
}
```

4.1.3 원격 태스크 구문

원격 태스크 구문의 구현시에 고려하여야 할 부분은 크게 두가지가 있다. 첫째, 분산 메모리간에 작업을 분산시키는 방법의 설계이다. 이것은 태스크로 정의한 병렬 실행 단위가 독립 주소 공간을 가지기 때문에 다른 노드의 태스크 생성, 태스크의 실행을 직접 제어할 수 없기 때문이다. 둘째, 태스크간의 자료 전송은 함수의 매개 변수 형태로 이루어지는데, NORMA 구조의 시스템에서는 지정된 자료가 직접 전달되지 못하므로, 분산 메모리 환경에 위치한 다른 태스크로 자료가 전송되도록 하여야 한다.

이러한 문제점을 해결하기 위해 사용한 방법은 다음과 같다. 첫째, main 함수를 가진 파일은 주태스크로 지정하고 task 구문으로 정의한 함수를 가진 파일은 부태스크로 지정한다. 주태스크는 부태스크의 실행을 제어하기 위한 초기값의 송신과 부태스크의 초기화 결과의 수신을 담당하고, 부태스크는 주태스크에 의해 실행이 시작되어, 주태스크에서 전송한 부태스크 함수의 호출을 인지하여 함수를 실행하고, 실행 결과를 전달하는 부분을 담당한다.

태스크의 실행 위치는 태스크/노드 구성 파일에서 지정된다. 이러한 주태스크와 부태스크의 실행이 이루어지도록 하기 위해 실행시간 라이브러리를 구현하였으며, 이것은 UNIX의 프로세스와 소켓을 기반으로 구현되었다. 이것은 통신 환경을 UNIX의 기본 기능을 이용함으로써 다른 시스템에서도 동작할 수 있도록 하기 위함이다.

둘째, 태스크간의 자료 전송을 위한 기반 환경은 포트를 사용하였으며, 전송될 자료는 struct 타입으로 패킹되어 포트를 통하여 전송되도록 하였다. 자료의 수신은 블록킹(blocking) 방법을 사용하도록 구현하였다.

(그림 8)은 지금까지 설명한 방법을 적용하여 (그림 6)의 ParaC 프로그램을 번역한 예이다. (그림 6)의 각 프로그램은 태스크 실행을 위한 실행시간 라이브러리로 번역되며, 실행시에 태스크로 실행된다. (그림 6)의 C1, C2, C4 코드는 각각 C'1, C'2, C'4로 변환되며, 부태스크의 C6, C7, C8 코드는 C'6, C'7, C'8로 변환된다. 여기에서 C'3 코드는 C3의 parallel 구문이 번역된 부분이다.

```
/* 번역된 주태스크 프로그램: main.c */

typedef struct {
    _TASK cid;
    int n;
} _PRODUCER_1;
_PRODUCER_1 _producer_1;

int __call_tcall_producer_1 (_TASK task, char
*task_name, char *params, int size)
{
  int _result;

  _send_task_name (task, task_name);
  _send_params (task, params, size);
  _receive_result (task, (char *) &_result, sizeof
(_result));
  return (_result);
}
..... 중략 .....

void                                    /* C'3 */
_par_thr_fun_1 (_par_thr_argt_1 * _par_thr_arg_1)
{ {
    _producer_1.cid = (*(_par_thr_arg_1->cid));
    _producer_1.n = (*(_par_thr_arg_1->n));

    __call_tcall_producer_1 ((*(_par_thr_arg_1->pid)),
"   producer ", (char *) &producer_1, sizeof
(_PRODUCER_1));                         /* C'4 */
  } }
..... 중략 .....

main (int argc, char *argv[]) {
  _TASK pid;
  _TASK cid;
  int n = 1;

  _lookup_taskgroup ();
  pid=_call_tcreate("/home/parac/producer"); /* C'1 */
  cid=_call_tcreate("/home/parac/consumer"); /* C'2 */
  {
    ..... 중략 .....

    error = pthread_create (&_par_thr_1, NULL, (void
*(*)()) _par_thr_fun_1, &_par_thr_arg_1);
    error = pthread_create (&_par_thr_2, NULL, (void
*(*)()) _par_thr_fun_2, &_par_thr_arg_2);

    ..... 중략 .....

    pthread_join (_par_thr_1, NULL);
    pthread_join (_par_thr_2, NULL);
  }
  _terminate ();
}

/* 번역된 부태스크 프로그램: producer.c */

typedef struct {
    int send_data;
} _PARAMS_1;
_PARAMS_1 _params_1;
typedef struct {
    _TASK cid;
    int n;
} _PRODUCER_2;
_PRODUCER_2 _producer_2;

main (int argc, char *argv[]) {
  int producer (_TASK, int);
  char _temp_name[128];
  int _stringlen;  _TASK cid;  int n;

  _port = _get_port ();
  _mytaskid = _send_task_desc (_port, argv[1]);
  _stringlen = strlen (" producer ");
  while (1) {
      _receive_task_name (_port, _temp_name);
      if (!memcmp (_temp_name, " producer ",
_stringlen)) {
          if (_receive_params ((char *) &_producer_2,
sizeof (_PRODUCER_2)) < 0)
            fprintf ((&_iob[1]), "pcc: Fail : Received
parameters\n");
          cid = _producer_2.cid;
          n = _producer_2.n;
          producer (cid, n);
          {
            int _result;

            _result = (1);
            _send_result (_mytaskid, (char *)
&_result, sizeof (_result));
          }
        }
      if (_down (_temp_name))
        _shutdown (_mytaskid);
    }
}
producer (_TASK cid, int n) {                /* C'6 */
  int i;
  static int send_data = 100;
  for (i = 0; i < n; I++) {
      _params_1.send_data = send_data;
      _send (cid, (char *) &_params_1, sizeof
(_PARAMS_1));                            /* C'7 */
  }
}

/* 번역된 부태스크 프로그램: consumer.c */

..... 중략 .....

main (int argc, char *argv[]) {
  ..... 중략 .....
        consumer (pid, n);
  ..... 중략 .....

}
consumer (_TASK pid, int n)                  /* C'8 */
```

```
{
  int i;
  static int receive_data;

  for (i = 0; i < n; I++) {
      _receive (pid, (char *) &_params_1, sizeof
(_PARAMS_1));                                /* C'9 */
      receive_data = _params_1.receive_data;
  }
}
```

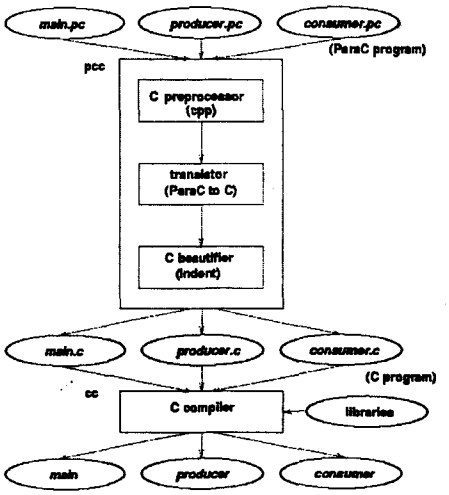
(그림 8) 번역 예
(Fig. 8) An example of the translation

5. 구현 시스템의 시험 및 비교

5.1 구현 시스템의 시험

구현된 번역 시스템은 ParaC 번역기와 실행시간 라이브러리로 구성되어 있다. 번역기는 C로 구현되었으며, lex를 사용한 어휘 분석기(scannar)와 yacc을 사용한 구문 분석기(parser)로 되어 있다. 그리고 변수를 처리하기 위한 심볼 테이블은 해쉉 방식을 사용하였다[8].

입력된 프로그램은 (그림 9)와 같이 C 프리프로세서에 의해 처리된 후 번역되도록 하였으며, 출력된 C 코드는 C 미화기를 사용하여 정리되도록 하였다. 번역된 코드는 링커에 의해 실행시간 라이브러리와 연결되도록 하였으며, 실행시간 라이브러리는 C로 구현하였다. 여기에서 원격 태스크 구문의 실행을 위한 태스크 관련 라이브러리는 SPAX의 운영체제에서 제공하는 소켓을 사용하여 구현하였다.



(그림 9) 번역 시스템의 구조
(Fig. 9) Overview of the translation system

구현된 번역 시스템은 Pentium pro 프로세서가 4개 장착된 2대의 노드를 기반으로 하고 Chorus 마이크로커널과 UnixWare를 서버화한 병렬 운영체제가 탑재된 SPAX 테스트베드 시스템에서 시험을 완료하였다. 시험을 통하여 ParaC 언어가 효율적인 병렬 프로그래밍 환경을 지원하고, 번역된 코드가 분산 메모리 환경에서 정상적으로 실행됨을 확인하였다.

5.2 관련 연구와의 비교

본 논문과 유사한 개념으로 C 언어를 확장하는 방법을 사용하여 개발된 언어로는 Digital사에서 분산 메모리 시스템을 위해 개발한 Cid, IBM사에서 공유 및 분산 메모리 모델의 컴퓨터에서 사용될 수 있도록 설계한 pC, Parsec Development사에서 트랜스퓨터 컴퓨터에서 사용하기 위해 개발한 Par.C, 그리고 Berkeley 대학에서 분산 메모리 시스템을 위해 개발한 Split-C 등이 있다[9, 10, 11, 12].

그러나 본 논문에서 설계한 ParaC 언어는 다음과 같은 부분에서 앞서 기술한 언어와는 다른 장점을 가지고 있다.

첫째, ParaC 언어는 확장성 높은 병렬 프로그래밍 환경을 지원한다. 확장성을 고려하여 개발된 언어는 Cid나 Par.C등 다수개가 존재하지만 대부분의 언어는 병렬화의 단위를 제한하여 제공하는 구조로 되어 있다. 병렬화의 단위를 쓰레드만으로 지원하는 언어의 경우, 병렬화를 fine grain이나 medium grain만으로 고정하여 제공하는 제약을 가지며, 실제 실행 환경에서는 쓰레드의 갯수가 일정한 갯수에 도달하면 프로그램의 성능이 저하되는 문제점이 있다.

둘째, 기존의 C 언어와 호환되는 문법을 제공한다. Par.C나 pC등 기존의 병렬 언어는 병렬성을 지원하기 위해 C 언어의 장점인 포인터 변수의 사용을 제한하고 병렬 구문의 중첩 표현을 금지하는 구조로 되어 있다. 이것은 C 언어의 표현력을 제한하고 병렬 표현을 단순화하는 단점이 된다.

셋째, 일반 사용자에게 쉬운 병렬 구문을 제공한다. Cid나 Split-C같은 기존의 병렬 언어는 라이브러리 형태의 병렬 구문을 제공하거나, 제공된 병렬 구문을

사용하기 위해서는 구문을 사용하기 전에 많은 제어 구문을 사용자가 명시하여야 하는 복잡한 형태로 되어 있다.

6. 결 론

본 논문에서는 공유 및 분산 메모리 구조를 가진 병렬 시스템의 병렬 프로그래밍 환경을 효과적으로 지원하기 위하여 ParaC 언어를 설계하고 구현한 내용에 대하여 기술하였다. ParaC 언어는 C 언어에 병렬 구문을 추가하여 확장성 높은 병렬 시스템에서 프로그래밍을 효과적으로 할 수 있도록 설계되었으며, 설계는 시스템에 적합한 프로그래밍 모델을 정의하고 기존의 C 문법과 호환되는 구조의 병렬 구문을 추가하는 방법으로 이루어졌다. 또한 ParaC 번역 시스템을 구현하기 위해 고려하여야 할 주요 문제점을 정의하였고, 이를 해결하기 위한 방안을 기술하였다. 그리고, ParaC 번역기와 분산 환경에서의 병렬 실행에 필요한 실행시간 라이브러리를 구현하였으며, 구현된 번역 시스템은 두 개의 GN 노드로 구성된 SPAX 테스트베드 시스템에서 시험을 완료하였으며, ParaC 언어로 작성된 코드가 효과적으로 시스템의 분산 환경을 표현함을 확인하였다.

추후에 SPAX 시스템이 안정되면 ParaC 번역 시스템의 성능 평가를 실시할 예정이며, 쓰레드 및 태스크의 수행 성능을 최적화하기 위한 방법에 대하여 연구가 이루어질 예정이다. 또한, 본 논문에서 개발된 ParaC 언어는 UNIX 환경을 제공하는 다른 병렬 시스템에 탑재하는 연구가 이루어질 예정이다.

참 고 문 헌

[1] George S. Almasi and Allan Gottlieb, 'Highly Parallel Computing', 2nd ed., pp. 227-229, The Benjamin/Cummings Publishing Company Inc., 1994.

[2] Y. W. Kim, S. W. Oh, and J. W. Park, "Design Issues and the System Architecture of TICOM-IV, A Highly Parallel Commercial Computer," Proc. of Euromicro Workshop on Parallel and Distributed Processing, pp. 219-226, Jan. 1995.

[3] Kai Hwang, 'Advanced Computer Architecture: Parallelism, Scalability, Programmability', pp. 138-149, McGraw-Hill, Inc., 1993.

[4] ANSI, ANSI Programming Language C, Document No. X3.159-1988, ANSI, 1989.

[5] K. Mani Chandy and Carl Kesselman, CC++: A Declarative Concurrent Object Oriented Programming Notation, California Institute of Technology, 1993.

[6] A. V. Aho, R. Sethi, and J. D. Ullman, 'Compilers, Principles, Techniques, and Tools', pp. 279-336, Addison-Wesley Publishing Company Inc., 1988.

[7] Frank Mueller, "A Library Implementation of POSIX threads under UNIX," Proc. of the USENIX Conference, pp. 29-41, Jan. 1993.

[8] C. W. Fraser, and D. R. Hanson, 'A Retargetable C Compiler: Design and Implementation', pp. 35-52, The Benjamin/Cummings Publishing Company, Inc., 1995.

[9] R. S. Nikhil, "Cid: A Parallel "Shared-memory" C for Distributed Memory Machines," in Proc. 7th. Ann. Workshop on Languages and Compilers for Parallel Computing, pp. 376-390, Springer-Verlag LNCS 892, 1994.

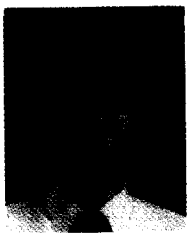
[10] Canetti, L. P. Fertig, S. A. Kravitz, D. Malki, R. Y. Pinter, S. Porat, and A. Teperman, "The parallel C(pC) Programming Language," IBM J. Res. Develop., Vol. 35, No. 5/6, Sep./Nov., pp. 727-741, 1991.

[11] Parsec, 'Par.C System: User's Manual and Library Reference', Parsec Developments, 1990.

[12] D. E. Culler, A. Dusseau, S. C. Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, K. Yelick, "Parallel Programming in Split-C," Proceedings of Supercomputing '93, Portland, Oregon, pp. 262-273, Nov. 1993.

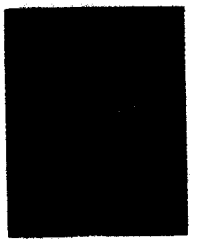
이 경 석

1990년 광운대학교 컴퓨터공학과 졸업(공학사)
1992년 광운대학교 대학원 컴퓨터공학과 졸업(공학석사)
1992년~현재 한국전자통신연구원 연구원
관심분야:프로그래밍 언어, 병렬 컴파일러, 멀티미디어



우 영 춘

1986년 경북대학교 전자공학과 졸업
1988년 경북대학교 대학원 전자공학과 졸업(공학석사)
1988년~현재 한국전자통신연구원 선임연구원
관심분야:분산 계산 기술, 병렬 프로그래밍 언어, 멀티미디어



김 진 미

1988년 부산대학교 계산통계학과 졸업(학사)
1997년~현재 충남대학교 컴퓨터과학과 석사과정 재학중
1988년~현재 한국전자통신연구원 선임연구원
관심분야:병렬 컴파일러, 컴파일러 최적화, 병렬 처리

지 동 해

1982년 경희대학교
1986년 Iowa State University (공학석사)
1988년 Iowa State University (공학박사)
1988년~현재 한국전자통신연구원 병렬프로그래밍 연구실장
관심분야:병렬 및 분산처리, 객체지향 프로그래밍 환경, 멀티미디어