DP를 이용한 병렬 프로그래밍에 관한 연구

°오 병 균* 박 찬 정** 신 인 철** 이 상 범**
*목포대학 전산통계학과 **단국대학교 전자공학과

A Study on the Distributed Processes for Concurrent Programming

Byeong-Kyun Oh * Chan-Jung Park ** In-Chul Shin ** Sang-Burm Rhee **
* Mog-Po National University ** Dan-Kook University

ABSTRACT

This paper introduces distributed processes — a new language concept for concurrent programming.

It is proposed for real-time applications controlled by computer network with distributed storage.

These processes communicate and syschronize by means of procedure calls and guarded regions.

The paper gives several examples of distributed processes.

I. 서 론

오늘날 Computer의 활용 범위가 광범위 해지고 설계 기술도 발달함에 따라 Computer System의 구성 방법도 중앙 집중 방식에서 분산체제에 의한 Computer network의 활용이 보편화 되고 있다. 또한 이를 효율적으로 활용하기 위한 O/S분야의 연구 방법으로서 병렬 프로그래밍 (Concurrent Programming) 기법에 관한 많은 연구가 이루어 지고 있다[1,8]

Concurrent Programming 기법이란 여러개의 process들이 동시에 존재하여 서로 협력하거나 독립적으로 주어진 Job을 처리하는 program을 말하며, 여러 process가 서로 협력하여 주어진 Job을 의도한대로 수행하기 위해서는 어떠한 형태로든지 정보교환 (Communication)이 이루어져야 하고 또 정보교환 요구시 반드시 동기화 (Synchronization)가 가능해야 한다. [5,7]

본 연구에서는 Concurrent Programming을 위한 새로운 언어 개념으로 DP (distributed processes)를 소개하고자 한다.[1,2,4]

DP는 분산기억 방식에 의해 Computer network로 제어되는 real-time 응용에 적합한 개념이며, 본 논문에서는 DP를 이용하여 Procedure, Processes, Semaphores, Buffer …등에 적용해 보고자 한다. [2,7]

real-time 응용이란 real-time의 제한에 관한 computer와 progrmming 기법을 추구하는 것이며, real-time system은 임계 시간 제한 (critical timing constraints)에서 동시에 일어나는 activity들을 계속적으로 틀림없이 제어하기 위한 system이다. 이를 구현하기 위한 real-time program의 궁극적인 목표는 간편성, 신뢰성, 효율성 이므로 바람직한 real-time program을 작성하기 위해서 세부적이고 복잡한 것들은 추상적인 프로그래밍 언어로 작성하는 것이 바람직하다. [1,8]

language designer의 관점에서 볼 때 RTP(real-time program)는 다음과 같은 특성을 갖는다.

1) RTP는 여러가지 일들이 고속으로 동시에 일어나는 환경에서 interact 한다.

2) RTP는 1)의 환경으로 부터 다양한 nondeterministic request에 응답해야 하며, 이들 request가 형성된 순서는 예측할 수 없지만, 한정된 시간 안에 응답되야지 그렇지 않으면 입출력 data는 의미를 상실하게 된다.

3) RTP는 processor와 주변 장치등 여러개의 자원으로 구성된 computer를 제어하며, 그러한 환경에서 concurrent task의 여러가지 일들을 실행한다.

4) RTP는 computer가 작업을 하고 있는한, 그 환경을 계속 제공해야 한다.

real-time 응용에서 요구되는 것은 nondeterministic request에 신속히 응답할 수 있는 많은 것들 중에서 concurrent task를 지정할 수 있는 능력이 있어야 한다.

concurrent pascal과 modula라는 programming language는 abstract concurrent programming에 대한 요구를 만족하고 있으며, 이들은 monitor 개념에 기초를 두고 있다. 특히 modula는 근본적으로 single processor에서의 multiprogramming 중심이다. concurrent pascal의 직접적인 구현을 위해서는 공유기억에 의한 single process나 multiprocessor가 필요하다. 표현 형식에 있어서 이

들 언어는 분산기억의 microcomputer network에 대한 이론이 아니므로 분산기억의 제한(constraints)을 만족하기 위해서는 concurrent pascal을 수정해야만 하는데 이러한 착상에 의해 monitor와 process 개념을 통합하는 것이 바람직하다.

real-time 응용을 위한 새로운 언어 개념은 다음과 같은 특성을 갖는다.

1) RTP는 동시에 시작되는 병행 process들로 구성되고 항상 실행 가능해야 한다.

2) process는 다른 process내에 정의된 common procedure를 call할 수 있고, 이 procedure는 다른 process가 어떤 조건에 대해 true가 되기를 기다리고 있을 때 실행할 수 있으며, 이것이 process communication의 형식이다.

3) procedure들은 guarded region[4]이라 부르는 nondeterministic statement에 의해 동기화 된다. 이들 process는 공유나 분산기억의 multiprocessor system에서 program module로 사용될 수 있으며, real-time 제한을 만족하기 위하여 각 processor는 single process를 지정해야 한다. 또한 process가 true가 될 어떤 조건을 기다릴 동안 그 processor는 또다른 external procedure call을 기다린다. 이는 자원의 낭비가 아니라 해당 processor에 대한 처리할 Job의 일시적 결핍을 나타낸 것이며, process들 사이의 parameter passing은 공유기억의 경우는 copying에 의하고, 분산기억의 경우는 I/O에 의하여 communication된다. [5,7]

Ⅱ. DP 언어의 개념

concurrent program은 동시에 실행할 여러개의 sequential process들로 구성된다.

여기서는 이들 여러 process들 사이에 communication과 synchronization 문제를 해결하고, concurrent programming 기법을 real-time 응용에 효율적으로 적용할 수 있는 DP라는 새로운 언어 개념을 알아보자.

1) process는 own variable, some common procedures, 및 initial statement로 정의된다.

```
process name
own variable
common procedures
initial statement
```

그림 1. procedure의 구조

process는 오직 own variable에 의해 access되고 공유변수는 없으며, 자신이나 다른 process에 정의된 common procedure를 call할 수 있다. 이때 어떤 process로 부터 다른 process의 procedure call을 external request라 한다.

process는 initial statement와 external request의 두 종류 operation을 수행하는데, 이들 operation은 interleaving에 의해 한번에 하나씩 수행한다.

initial statement를 수행하므로서 시작되는 process operation은 statement가 끝나거나 true일 조건을 기다릴 때까지 계속되며, external request에 의해 시작되는 operation은 이 operation이 끝나거나 어떤 조건을 기다릴 때 다른 process에 의해 request된 또다른 operation을 시작하거나, true가 된 조건의 결과에 의해 이전의 operation을 실행한다.

이와 같이 initial statement가 끝나더라도 process는 계속 존재하여 external request를 받아 들일 수 있어야 한다.

따라서 interleaving은 program에 의해 제어되며, process는 operation이 끝나거나 guarded region내에서 어떤 조건을 기다릴 때, 하나의 operation으로 부터 다른 operation으로 교대(switch) 된다.

process는 모든 현행 operation들이 guarded region내에서 delay될 때나 다른 process를 request할 때 이외에는 operation 수행을 계속하는데, 전자는 process가 다른 process로 부터 자신을 call할 때까지 idle하고, 후자는 다른 process에 의해 request된 operation을 완료할 때까지 idle한다.

process는 진행해야 할 operation이 있는한 그 수행을 보장해야 하고, programer는 모든 operation이 한정된 시간 안에 끊임없이 수행되도록 보장해야 한다.

2) procedure는 I/O parameter, local variable, call됐을 때 수행할 statement로 정의된다.

```
proc name ( input para # output param )
local variables
statement
```

그림 2. procedure의 구조

process P는 다음과 같이 다른 process안에 정의된 procedure R을 call 한다.

```
call Q.R ( expressions, variables )
```

이 때 R이 수행되기 전에 call 안의 expression value는 input parameter에 의해 할당되며, process 사이의 parameter passing은 공유기억의 경우는 copying에 의하고, 분산기억의 경우는 I/O에 의해 구현된다.

본 연구에서 process는 어떤 제한없이 다른 process내의 procedure를 call할 수 있는 것으로 전제한다.

3) nondeterminism은 guarded command와 guarded region이라고 부르는 두가지 statement에 의해 제어되며 guarded region은 operation을 delay할 수 있으나 guarded command는 그렇지 않다. guarded command는 그 변수의 현재 상태를 조사하여 process가 여러 statement들 중에서 하나를 선택하여 실행할 수 있게 한다.

guarded command는 다음과 같은 syntax와 meaning을 갖는다.

if B1 : S1 | B2 : S2 | …… end

do B1 : S1 | B2 : S2 | …… end

그림 3. guarded command의 구조

4) guarded region은 그 variable의 상태가 여러 statement들 중에서 어느 것을 선택할 수 있을 때까지 waiting하며, 여러 alternative중의 어느 것도 현재 상태에서 실행 가능한 것이 없으면 process는 guarded region의 수행을 delay한다.

guarded region은 다음과 같은 syntax와 meaning을 갖는다.

when B1 : S1 | B2 : S2 | …… end

cycle B1 : S1 | B2 : S2 | …… end

그림 4. guarded region의 구조

5) data type은 integer, boolean, character이거나 어떤 type T의 n개의 원소로 이루어진 set, sequence, array이며, 그 형식은 다음과 같이 표시한다.

int bool char

set[n]T seq[n]T array[n]T

그림 5. data type의 표현 형식

6) 다음 statement는 data structure에서 모든 element들을 나열하는 방법이다.

for χ in y : S end

그 밖에 empty statement는 skip을 나타내고, semicolon(;)은 optional이다.

Ⅲ. process의 communication

앞에서 정의한 distributed process의 언어 개념을 concurrent programming에 적용한 몇가지 예를 통하여 procedure call에 의한 communication을 알아보기로 하자. [9,10]

o Semaphore의 예

zero로 초기화된 일반적인 semaphore는 wait와 signal operation에 의하여 sem이라는 process로서 다음과 같이 구현할 수 있다.

```
PROCESS  sem : s : int
PROC  wait WHEN s>0 : s:=s-1 END
PROC  signal : s:=s+1
s:=0
```

initial statement는 semaphore에 zero값을 할당하면 끝나지만, 이 process는 계속 존재하여 다른 process들에 의해 call될 수 있는데, call하는 형식은 다음과 같다.

CALL sem.wait CALL sem.signal

o Message Buffer의 예

buffer process는 send와 receive operation에 의해 process간에 전송되는 일련의 문자들을 저장한다.

```
PROCESS  buffer ; S:seq[n] char
PROC  send ( c : char ) when not s.full : s.put(c)
      end
PROC  rec ( #y : char ) when not s.empty : s.put(y)
      end
S:=[ ]
```

initial statement를 시작하기 위해서는 buffer를 비워야 하며, buffer operation은 다음과 같이 call한다.

CALL buffer.send[χ] CALL buffer.rec[y]

o Resource Scheduler의 예

많은 user process들은 scheduling process내의 call request와 release operation에 의해 abstract resource를 exclusive access할 수 있어야 한다.

```
PROCESS  resource ; free : bool
PROC  request when free : free := false end
PROC  release if not free : free := true end
free := true
```

CALL resource.request …… CALL resource.release

여기서 boolean free의 사용은 request와 release operation의 alternation을 보다 강력하게 실행하기 위한 것이다.

Ⅳ. Implementation의 조건

다음은 DP의 implementation시 알아야 할 몇가지 일반적인 특성을 요약한 것이다.

1) 잘 design된 concurrent program에서는 각 process가 오직 이웃하는 소수의 process들끼리 communication하는 것으로 가정한다.

2) 각 processor는 DP의 P와 process P를 호출 할

DP들을 대표(representative)한 소수의 anonymous process를 갖는다.

3) processor가 idle일 때, 다른 processor로 부터 input data의 request를 받을 때까지 wait한 local representative를 activate한다. representative는 available input로 request된 local procedure를 call한다. procedure가 종료됐을 때, 출력 data가 다른 processor에 return되고, representative는 다시 passive된다.

4)두 processor간의 parameter passing은 procedure가 실행되기 전에 single input과 그것이 종료 됐을 때 single output을 요구한다.

5) single processor내에서 process switching 속도는 그 real-time 응답에 의해 결정된다.

V. 결 론

본 연구에서는 분산기억 방식인 multiprocess network에 DP를 적용하므로서 procedure call에 의해 process의 효율적인 communication 수행과 guarded command와 guarded region에 의해 nondeterminism에 대한 synchronization 문제를 해결할 수 있음을 알았다. 이 과정에서 concurrent programming을 위한 DP의 장점으로는,

1) DP는 class, monitor, process 개념을 통합한 process, procedure, conditional critical region의 결합이다.

2) DP는 procedures, classes, monitors, processes, semaphores, buffers, input/output와 같은 기본적인 programming 개념들을 포함하고 있다.

3) DP의 programming 개념들은 공통성을 갖기 때문에 각 개념에 대한 공통적인 증명 방법을 개발 할 수 있다.

4) concurrent pascal은 class, monitor, process에서 여러 개의 virtual instruction으로 분류되나, DP에서는 몇 개의 요소로 축소되며, 특별한 경우는 compiler에서 사라진다.

5) DP는 concurrent pascal에서 널리 알려진 concurrent 문제나 새로운 문제에 대해 보다 효율한 algorithm을 작성할 수 있게 해주며, 특히 공유변수가 없는 concurrency 문제와 single process에서의 nondeterminancy에 대한 요구를 잘 처리할 수 있다.

6) process간의 data 전송은 communication 기법인 guarded input command에 의해 자동적으로 동기화되며, 이 때 두 communicating process 사이의 관계는 대칭적이다.

7) DP는 network를 위한 실질적인 언어의 시도이며, 하나의 operation으로 input과 output의 변화가 빈번하다. 이 때 두 communicating process 사이의 관계는 비대칭적이며 이러한 관계는 계층 system에서 유용하다.

앞으로 이러한 새로운 언어 개념에 의한 증명 방법과 network 구조 등에 더 많은 연구가 요망된다.

참 고 문 헌

1) Andrews, G.R., Schneider, F.B., " Concepts and Notations for concurrent programming ", com. survey, vol. 15, No.1, pp. 3-43. 1983

2) Hansen, P.B., " Distributed process : A concurrent programming concept ", commun. ACM, vol. 21, No.11, pp. 934-941, 1978

3) Hansen, P.B., Operating system principle, Prentice-Hall, Cliffs, N.J., 1973

4) Dijkstra, E.W., " Guarded commands, nondeterminacy, and formal derivation of program," Comm. ACM, vol.18, No.8, pp. 453-457, 1975

5) Hoare, C.A.R., " Communicating Sequential Process," Commun. ACM 21, 8, pp. 666-677, 1978

6) Peterson, L., Silberschatz, A., Operation System Concepts, Addison-Wesely. Co., 1985

7) Shatz, S.M., " Communication Mechanisms for Programming Distributed Systems," IEEE Computer, vol. 17, No.6, pp. 21-28, 1984

8) Ben-Ari, M., Principles of Concurrent Programming, Prentice-Hall, Cliffs, N.J., 1982

9) 김길창, 김영수, "병렬 프로그래밍을 위한 동기화 구조," 정보과학회지, 제 4권, 제 1호, 1986

10) 이상범, "분산처리를 위한 프로그래밍 언어" 대한전자공학회, 제 3권, 제 1호, 1986