# 병렬 프로그램 성능 감시 도구의 성능 분석층에 관한 연구

김 병 기† · 마 대 성††


## 요 약

본 논문에서는 병렬 프로그램의 다양한 성능 분석을 위해 사건 표현식을 이용한 성능 분석층을 설계하였다. 사건 표현식은 프로그램의 동적 실행 상태를 나타내는 사건을 분석하기 위해 일반적인 프로그램 언어와 비슷한 형태로 구성하였다. 사건 표현식은 오버로딩 기능과 성능 분석에 필요한 필터 연산, 데이터 형식 변환 함수, 통계 함수 등 연산자와 함수들을 제공한다.

사건 표현식을 이용하여 프로그래머는 관심 있는 성능 분석을 위하여 사건 추적 데이터를 쉽게 가공할 수 있고, 기존의 도구보다 다양한 분석을 쉽게 할 수 있다.


# A Study On Performance Analysis Layer of Parallel Program Performance Monitoring Tool

Byung-Ki Kim† · Dai-Sung Ma††


## ABSTRACT

This paper designs the performance analysis layer for the various performance analysis of parallel programs using event expressions. The event expressions are similar to the normal program language to analyze the events which display a dynamic state exchange of a program. The event expressions suggest operations for overloading and functions which are needed in performance analysis, such as a filtering operation, data format translation functions, performance analysis, static functions, and etc.

By using the event expressions, the programmer can modify the event trace data to analyze the performance and analyze more easily and variously than the pre-developed tools.


## 1. 서 론

최근, 병렬 컴퓨팅에 대한 관심이 높아감에 따라 여러 가지 구조를 갖는 다양한 병렬 컴퓨터들이 개발되어지고 있다. 병렬 컴퓨터에서 응용 프로그램의 효율을 높이기 위해서는 컴퓨터의 구조에 맞는 프로그래밍 모델을 사용해야 한다. 응용 프로그래머는 서로 다른 컴퓨터 구조와 프로그래밍 환경에서 동작하는 복잡한 응용 프로그램들의 성능을 높이기 위해서 매번 프로그램을 수정해야하는 새로운 문제에 직면하게 되었다 [5,7,14].

이러한 문제점을 해결하기 위해 병렬 컴퓨터에서 동작하는 각종 응용 프로그램들의 성능을 높이기 위한

† 종신회원 : 전남대학교 컴퓨터정보학부 교수
†† 준 회 원 : 광주교육대학교 전산교육과
논문접수 : 1998년 12월 29일, 심사완료 : 1999년 3월 11일

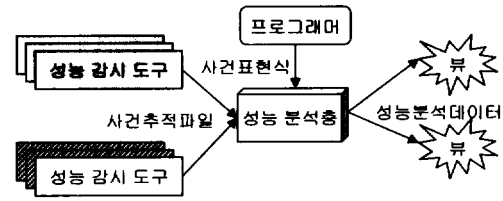
여러 가지 도구들이 개발되고 있다. 특히, 병렬 프로그램이 수행될 때 발생하는 각종 사건들을 추적하여 분석해주는 병렬 프로그램 성능 감시 도구에 대한 연구는 매우 활발히 이루어지고 있다[1,2,4,6,8,9,10]. 대부분의 성능 감시 도구는 병렬 프로그램의 성능 분석에 필요한 사건 추적층과 사건 추적층에서 수집된 데이터로부터 성능 분석 데이터를 필터링하여 추출하는 성능 분석층, 그리고, 필터링된 데이터를 이용하여 프로그래머가 이해하기 쉬운 형태로 나타내어 주는 가시화층으로 구성된다[1].

일반적으로, 사건 추적층에서는 병렬 프로그램의 성능에 영향을 미치는 사건을 정의하고, 어떤 종류의 사건 데이터를 수집할 것인가에 따라 사건을 추적하게 되는데, 사건의 범위에 따라 추적된 사건 데이터의 양은 매우 방대해진다. 사건 분석층에서는 사건추적층에서 수집된 데이터 중에서 프로그램의 성능 분석에 필요한 사건들만을 추출하고, 필터링하는 과정을 거치게 된다. 가시화층에서는 실제적이고 효과적으로 성능 분석에 도움을 줄 수 있도록 가시화(Visualization)를 제공한다[1,5,12].

ParaGraph[1]은 메시지 전달 프로그램을 분석할 수 있도록 개발되어 메시지 전달 프로그래밍 모델에 관련된 분석 결과를 알려준다. Pablo[2,9]는 성능 감시 도구들의 특징과 기능들을 통합하여 병렬 프로그램의 실행시 발생하는 정적 정보를 시각화하여 보여준다. 병렬 프로그램의 성능 분석을 위해 SDDF 형태[13]의 성능 분석 데이터를 이용하여 프로그래머는 분석 모듈에서 제공된 분석 방법에 의하여 성능 분석과정을 거친다.

대부분의 성능 감시 도구들은 병렬 컴퓨터의 특성상 특정 시스템 또는, 극히 제한된 시스템을 위한 도구로서 개발되었다. 특정한 시스템에서 개발된 성능 감시 도구들은 시스템에 의존적인 성능 분석 결과만을 제공함으로서 분석의 다양성과 도구의 확장성이 부족하다는 단점을 가지고 있다. 물론 사건 추적층은 하드웨어에 의존적이어서 서로 다른 시스템에 이식하여 성능 분석 데이터를 추출한다는 것은 거의 불가능한 일이다. 이러한 이유로 현실적으로 범용성 있는 성능 감시 도구를 개발한다는 것은 매우 어려운 일이다. 그러나, 성능 분석층과 가시화층은 성능 감시 도구의 사건 추적층에서 추출되는 사건 데이터와의 인터페이스만 잘 정의된다면 독립적으로 개발이 가능하다.

본 논문에서는 범용성 있는 병렬 프로그램의 성능 감시 도구 개발을 위해 성능 분석층을 설계하고 구현하였다. 본 논문에서 제안한 성능 분석층은 사건 기반형 성능 감시 도구[1,2,10]들에서 발생하는 사건 추적 데이터들을 접근하여 다양한 성능 분석을 할 수 있도록 일반적인 프로그램 언어와 비슷한 형태의 수식으로 표현해 주는 사건 표현식을 제공한다. 프로그래머는 본 논문에서 제안한 사건 표현식을 이용하여 분석하고자 하는 목표에 맞게 사건 추적층에서 추출된 사건 데이터를 분석하고 필터링한다. 사건 표현식은 오버로딩 기능과 성능 분석에 필요한 데이터 형식 변환 연산자, 통계 함수등 연산자와 함수들을 제공함으로써 프로그래머에게 다양한 분석 결과를 제공할 수 있다(그림 1).



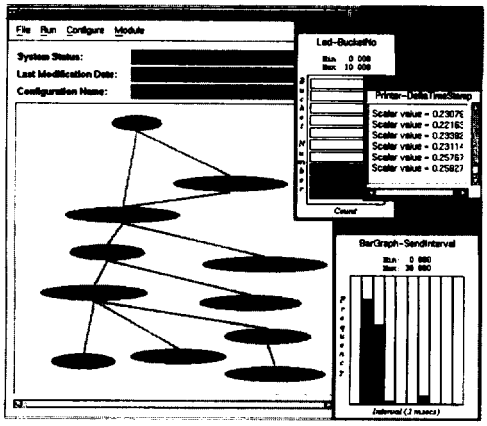

(그림 1) 제안한 성능 분석 과정

본 논문의 구성은 다음과 같다. 2장에서는 현재 개발 진행중이거나 개발되었던 성능 감시 도구들에 대해 알아보고, 3장에서는 사용자 정의 분석을 위한 성능 분석층의 구조에 대해 기술한다. 4장에서는 사건 표현식을 이용한 성능 분석 방법에 대해 기술하고 5장에서는 결론 및 향후 연구과제에 대해 논한다.

## 2. 관련 연구

병렬 프로그램의 성능을 분석하기 위해 개발된 대표적인 도구로는 Pablo[2,9], ParaGraph[1], AIMS[10] 등이 있다. Pablo는 일리노이 주립대학에서 개발한 도구로서 여러 가지 종류의 병렬 성능 감시 도구들의 특징과 기능들을 통합하여 병렬 프로그램의 실행시 발생하는 정적 정보를 그래픽으로 보여주도록 설계되었다. Pablo는 SDDF(Self Defined Data Format)[13]라는 프로그램 추적 데이터 양식을 통해 사건 데이터를 사용자가 정의할 수 있도록 하고 있다. 또한, 다양한 형태의 추적 양식들을 변환기를 통해 SDDF 양식으로 변환할 수 있도록 하고 있다. 또한, 분석에 필요한 데이터 필터링, 데이터 가공, 가시화의 절차들을 데이터 흐

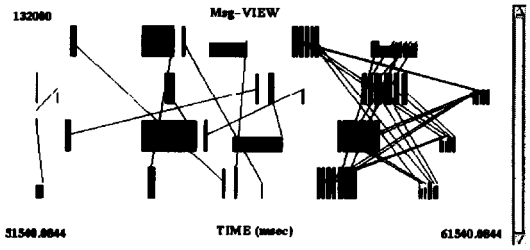
름의 측면에서 그래픽하게 표현하여 제공해 주고 있다. 가시화 층에서는 추상화된 뷰를 제공하므로써 분석의 다양성과 분석되는 데이터들간의 관계를 쉽게 이해하도록 하고 있다.




(그림 2) Pablo의 추적 분석 그래프

ParaGraph은 1990년 Oak Ridge National Laboratory에서 개발된 도구로서, 메시지 전달 컴퓨터상의 병렬 프로그램의 동작에 대한 동적인 그래픽 애니메이션을 제공하고, 실행 중에 사건 추적을 저장하고 나중에 분석하도록 하는 사후처리(post-moterm)를 제공한다. 또한, PICL(Portable Instrumented Communication Library)은 메시지 전달 병렬 컴퓨터 상에서 동작하는 프로그램들의 사건이 기록되어 있는 추적 파일을 생성할 수 있도록 한다. 추적 파일을 통한 성능 분석 결과는 각 디스플레이마다 내부 연산 모듈을 통해 결과를 나타낸다. ParaGraph은 다양한 메시지 전달 프로그램의 분석 정보를 제공하지만 메시지 전달 프로그램 외의 분석은 불가능하다. 또한 각 디스플레이마다 제공하는 정보가 이미 정해져 있어 다양한 관점의 분석이 불가능하다.

AIMS는 NASA Ames에서 개발한 성능 감시 도구로서 C와 Fortran 프로그램에 대한 성능 분석을 하는 분석도구이다. 메시지 전달 프로그램에 대한 사건 추적 기능을 제공하고 있고 프로그램의 성능 향상을 돕기 위해 Xinstrument 도구와 VK 도구, Tally로 구분되어진다. Xinstrument는 사건 추적층에 해당하는 도구로서 사용자가 사건을 추적할 특정 소스 코드 영역만을 선택하여 프로그램을 분석할 수 있도록 지원한다. 추적 영역이 설정되면 컴파일과 링킹 과정을 거쳐 추적 파일을 생성한다. 사건 추적 파일은 타임 스탬프와 함께 메시지 송신과 수신, 그리고 동기화 등을 기록한다. VK는 가시화층에 해당하는 도구로서 순서화된 추적 파일을 이용하여 애니메이션 한다. 사용자가 Xinstrument 도구를 이용하여 지정한 추적 정보들, 즉, 활성화된 서브루틴, 디스크 억세스, 메시지 송수신등의 정보를 가시화 뷰를 통해 보여주게 된다. 각각의 가시화 뷰는 시간에 의한 변화들을 보여주고, 내용에 따라 빨리 또는 정지되어 있는 정보들을 보여주도록 하고 있다.




(그림 3) AIMS VK의 메시지 뷰

그러나, 이와 같이 이미 개발되어진 도구들은 성능 분석을 위한 방법들이 제한되어져 있고, 특정 하드웨어에서만 동작을 한다는 단점을 가지고 있다. 또한, 프로그래머가 작성한 병렬 프로그램을 서로 다른 시스템에 적용하여 비교 분석한다는 것이 거의 불가능하였다. 본 연구에서는 이와 같은 문제점을 해결하기 위해 새로운 성능 분석층의 구조를 제안한다.

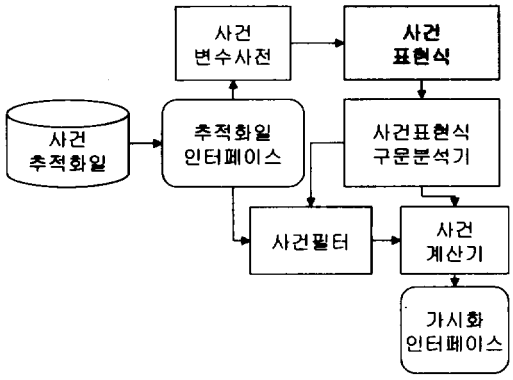
## 3. 사용자 정의 분석을 위한 성능 분석층의 구조

병렬 프로그램의 성능은 순차적인 프로그램과는 달리 예기치 못한 요인들에 의해 프로그램의 성능이 결정된다. 이러한 이유로 프로그래머가 병렬 프로그램의 성능을 미리 예측하고, 분석하는 것은 매우 어렵다. 이를 위해 성능 감시 도구는 프로그래머가 프로그램의 성능을 분석하고 수정할 수 있도록 충분한 분석 정보를 제공하여야 하지만, 기존의 성능 감시 도구는 특정한 시스템에만 동작하도록 개발되었기 때문에 새로운 형태의 시스템으로 이전하거나, 서로 다른 성능 감시 도구의 사건 추적 결과를 분석한다는 것은 거의 어렵다.

이러한 문제점을 해결하기 위해 본 연구의 성능 분

석층에서는 프로그래머가 임의의 성능 감시 도구에서 얻어진 사건 추적 데이터를 이용하여 프로그래머의 분석 목표에 맞는 분석 정보를 얻을 수 있는 방법을 제시하였다. 즉, 프로그래머는 병렬 프로그램의 성능을 분석하기 위하여, 본 논문에서 제공하는 사건 표현식을 직접 작성하여 사용자가 원하는 성능 분석 정보를 얻을 수 있도록 하였다.

이를 위하여 성능 분석층은 다음과 같은 구조로 구성하였다.




(그림 4) 성능 분석층의 구조

### 3.1 사건 변수 사전

사건 변수 사전은 사건추적파일에 기록되어 있는 추적된 사건 정보를 유지한다. 사건 변수 사전은 입력사건변수, 출력사건변수, 임시사건변수, 사건변수값, 플래그 등에 대한 자료구조를 유지한다. 사용자는 사건 변수 사전을 통해 입력사건변수와 사용자 정의형 임시사건변수를 검색할 수 있다.

| id | event name | event type | event value | flag |
|---|---|---|---|---|

(그림 5) 변수 사전의 자료구조

id는 입력사건변수, 출력사건변수, 임시사건변수의 식별자의 역할을 한다. event name은 사건 변수명이 기록되고, event type은 사건 데이터의 형을 가지고 있다. event value는 성능 데이터에 대한 실제 값을 가지고 있다. 플래그는 사건 분석 모듈에서 데이터가 갱신되면 1로 설정하고, 사건 계산기 모듈에서 계산이 끝나면 0으로 설정한다.

### 3.2 사건 표현식

사건 표현식은 사건 변수, 상수, 연산자, 함수들로 구성한 표현식이며, 일반적인 프로그래밍 언어에서의 표현식과 유사하다. 사건 표현식은 사건 추적 파일로부터 참조하고자 하는 사건을 분석하는데 이용된다. 프로그래머는 사건 표현식을 사용하여 참조하고자 하는 사건 이름, 사건 데이터의 통계, 사건 추적 파일내의 사건들을 조합하여 유도 될 수 있는 새로운 사건들을 기술할 수 있다.

#### 3.2.1 사건 표현식의 문법

사건 표현식에서 사용하는 문법은 일반 프로그래밍 언어와 유사하게 정의하여 프로그래머가 새로운 문법 형식을 익히지 않아도 쉽게 익숙해질 수 있도록 다음 (그림 6)과 같이 정의하였다.

```
expression : expression binary-op expression
           | unary-op expression
           | ( expression )
           | function
           | filtering
           | typecast
           | variable
           | constant
           ;
```

(그림 6) 사건 표현식의 문법

#### 3.2.2 사건 변수

사건 변수는 사건 표현식에서 쓰이는 기능에 따라 입력사건변수, 출력사건변수, 임시사건변수 등 세 가지 종류로 분류하였다.

입력사건변수는 사건 추적층에서 추적된 사건 추적 파일 내에 정의된 사건이름이다. 사건 추적 파일로부터 사건 데이터가 입력되면 해당 입력사건변수는 새로운 값을 저장한다.

출력사건변수는 가시화 뷰의 입력값을 저장하는 변수이다. 출력사건변수의 값이 새롭게 갱신될 때마다 출력사건변수의 변수값이 가시화 뷰의 입력값으로 전달되고, 가시화 뷰는 갱신된 입력값에 따라 디스플레이를 처리하도록 호출된다.

임시사건변수는 사용자가 사건을 표현하기 위해 표현식을 사용하여 임시로 정의한 사건 변수이다. 임시

사건 변수는 복잡한 사건 표현식을 분할하여 기술하거나, 사건 표현식 내에 자주 나타나는 부표현식의 중복 기술을 피하도록 할 수 있다.

사건 표현식에서 사건 변수는 최근의 데이터를 저장하고 있고, 사건 추적파일로부터 새로운 사건 데이터가 입력 될 때마다 새로운 값으로 바뀐다. 사건 표현식은 사건 변수의 값이 새로운 값으로 변할 때마다 다시 계산한다.

최종적으로 계산된 사건 표현식의 결과는 출력 사건 변수에 갱신되어 그 값은 가시화 뷰의 입력값으로 사용된다. 가시화 뷰는 호출되어 새롭게 갱신된 입력값에 따라 가시화 뷰를 갱신한다.

3.2.3 연산자 및 함수

사건 표현식에서는 사건 추적 파일내의 사건들을 계산하여 성능 분석 정보로 가공시킬 수 있도록 하기 위해 각종 연산자와 함수들을 제공한다. 사건 표현식에서 제공하는 연산자는 산술 연산자, 관계 연산자, 비교 연산자, 필터 연산자를 제공한다. 프로그램의 성능을 분석하기 위해 추출된 사건 데이터를 이용하여 특정한 조건인 경우, 예를 들어, 노드 n에서 노드 n+1로 보내지는 메시지들 중에서 특정 크기 이상의 메시지가 전송되는지 조사하고 싶을 경우 사건 표현식에서 제공하는 연산자들의 조합에 의해 조건에 맞는 사건들만을 추출해 낼 수 있다. 사건 표현식에서 제공하는 필터 연산자는 조건에 맞는 사건 데이터들만을 추출하고자 할 때 사용되는 연산자로서 <expression 2>의 조건이 참이면 <expression 1>의 사건이 추출되고, 거짓이면 <expression 1>의 사건 데이터는 무시된다. 나머지 연산자들도 일반 C 프로그램의 연산과 같다. 또한, 사건 표현식에서는 성능 분석에 필요한 각종 통계 데이터들과 성능 분석 데이터들을 다른 도구들의 도움 없이 성능 분석과정에서 쉽게 추출할 수 있도록 함수들을 제공함으로서 프로그래머가 성능 분석을 위해 사용하는 사건표현식을 간결하고 사용하기 쉽게 하였다. 현재 제공하고 있는 사건 표현식에 제공되는 연산자와 함수는 <표 1>과 같고 각 연산자들간의 우선 순위는 C 언어의 우선 순위에 따른다.

〈표 1〉 사건 표현식의 연산자

| | |
|---|---|
| 산술연산자 | +, -, *, /, % |
| 논리연산자 | !, &&, \|\| |
| 관계연산자 | ==, !=, <, <=, >, => |
| 비트연산자 | ~, &, \|, ^ |
| 비교연산자 | ?, : |
| 필터연산자 | <expression 1> if <expression 2> |
| 통계함수 | min(), max(), sum(), count(), avg(), var(), std() |

또한, 사건 표현식은 C++ 언어에서 지원되는 연산자와 함수에 대한 오버로딩(overloading) 기능[3]을 제공한다. 예를 들면, '+' 연산자에는 정수, 실수, 벡터, 배열, 문자열 데이터를 피연산자로서 허용한다. sum( ) 은 매개 변수로 정수, 실수, 벡터, 배열을 허용한다. 각 연산자와 함수의 리턴값의 데이터형식은 피연산자와 매개 변수의 데이터 형식에 따라 결정된다. 본 논문의 사건 표현식에서는 벡터와 배열에 대한 연산자/함수 오버로딩 기능을 별도로 추가하였다.

사건 표현식에서는 C 언어에서의 반복문에 해당되는 'for', 'while' 형식을 지원하지 않는 대신 배열, 또는 벡터의 원소들에 대한 파이프라인식의 연산기능을 제공하여 배열연산을 간단하게 표현할 수 있도록 하였다 (그림 7).

```
배열: A[7][7], B[7][7], C[7][7]

A = B + C <=> A[i][j] = B[i][j] + C[i][j], for i, j = 1...7
A = B + 1 <=> A[i][j] = B[i][j] + 1, for i, j = 1...7
A = sin(B) <=> A[i][j] = sin(B[i][j]), for i, j = 1...7
```

(그림 7) 연산자 오버로딩의 예

또한, 사건 표현식에서 어떤 연산자와 피연산자들 사이의 데이터형이 같지 않거나 어떤 함수의 매개 변수로서 요구되는 데이터의 형이 실제 매개 변수의 형과 일치하지 않을 경우 피연산자, 또는 매개 변수 값에 대한 데이터형 변환이 필요하다. 데이터의 형이 서로 맞지 않을 경우 구문 분석 단계에서 데이터형 변환 연산자가 자동적으로 삽입되어 강제적으로 데이터형을 변환하도록 하여 프로그래머가 사건 표현식을 쉽게 기술할 수 있도록 하였다.

3.3 사건 필터

사건 필터 모듈에서는 사건 표현식에서 사용되는 입력사건변수의 값을 추적 파일에서 사건계산기의 입력값으로 넘긴다. 구문 분석기로부터 사건 표현식에

사용된 입력 사건 변수에 대한 정보를 얻어 접근되어야하는 변수에 대한 정보를 얻는다. 입력 변수에 새로운 값이 갱신되면 사건 변수 사전의 해당 입력 변수에 플래그를 설정하고 사건 계산기를 호출한다.

### 3.4 구문 분석기

구문 분석기는 사용자에게서 입력되는 사건 표현식을 분석하여 우선 순위에 의한 단위 치환문들을 생성한다. 구문 분석기는 일반적인 컴파일러의 전단부 과정과 비슷한 형태의 과정을 거친다. 사건 표현식이 사용자에 의해 작성되면, 어휘 분석기(lexical analyzer)를 이용하여 토큰들을 구한다. 토큰은 사건변수, 예약어, 연산자로 구분되어지고 테이블에 정보를 유지한다. 사건변수는 사건 변수 사전의 변수들을 참조하여, 변수 사전에 등록되어 있는 변수는 입력사건변수로 테이블에 유지되며, 사건 변수 사전에 등록되어 있지 않은 사건 변수는 임시사건변수로 정의된다.

어휘분석과정이 끝난 후 파스 트리를 구성할 때에는 어휘분석단계에서 생성된 테이블을 참조하여 데이터 형식에 대한 정보를 얻은 후에, 구문분석단계에서 수식내의 데이터 타입이 다른 경우에 데이터 형식변환 연산자를 삽입한다. 데이터 형식변환 연산자는 사용자가 임의로 넣을 수도 있으나, 여기서는 사건 표현식내의 변수의 형을 비교하여 같은 형태로 강제 변환한다. 데이터 형식 변환연산자는 다음과 같이 정의한다.

<정의 1> 데이터 형식 변환 연산자

구문 분석 과정중 강제로 데이터형을 변환해주는 연산자를 데이터 형식 변환 연산자라 한다. 연산하고자 하는 두 데이터간의 형이 다른 경우 데이터형은 우선 순위에 따라 변환된다.

```
int N, float F
N + F => (float)N + F
            // 정수 N을 실수값으로 변환
N + (int)F  // 실수형 F가 정수 N으로 변환
                    (a)

int A[5];
A+1 => A+(int[5])1
            // 상수를 배열값으로 변환
    => A+(int[5]){1,1,1,1,1}
                    (b)
```

(그림 8) 데이터 형식 변환 연산자의 예

일반적으로 프로그램의 성능을 분석하기 위해 추출되는 사건추적파일에는 각각 타임스탬프와 해당사건에 대한 정보가 들어있다. 사건 추적파일내의 데이터는 타임스탬프의 순서대로 기록되므로 성능 분석층에서는 사건이 발생되는 순서에 따라 추적파일에 순차적으로 접근할 수 있다. 분석하고자 하는 사건은 해당 행위가 이루어질 때만 발생하므로 동기적으로 데이터가 발생하지는 않는다. 발생되는 사건 데이터에 의존하는 사건 표현식은 갱신되는 사건 데이터에 의존관계에 있는 수식만 계산하여야 한다. 물론 수식의 처음부터 계산과정을 되풀이 할 수 있으나, 이미 계산된 값을 다시 계산해야하는 등 시스템의 속도 저하를 가져오는 단점을 지니게 된다.

본 논문의 성능 분석층에서는 갱신되는 데이터와 의존 관계에 있는 수식만을 계산해내기 위해 데이터의 의존 그래프를 구성한다. 예를 들어, (그림 11)에서 'RECV' 값에 의존하는 식은 (2)번과 (3)번식이다. (1)번식은 'RECV' 변수의 갱신과 관계없으므로 계산될 필요가 없다.

<정의 2> 데이터 의존관계

변수가 갱신되었을 때 반드시 계산되어져야 할 수식이 있을 때 이를 데이터 의존관계에 있다라고 한다.

데이터 의존 관계에 따라 사건 표현식은 구문 분석기 내부에서 다시 단위 치환문으로 변환된다. 변환된 단위 치환문은 Topological Sort에 의해 우선 순위를 구한다. 각 단위 치환문은 연산 우선 순위가 설정되어 사건이 갱신되었을 때 그 사건과 의존관계에 있는 가장 높은 우선 순위의 단위 치환문부터 계산된다.

```
Algorithm for 구문 분석기
Input : 사건표현식
begin
    사건표현식으로부터 구문 트리 생성;
    // 데이터 형식 검사
    각 변수(노드)의 데이터 타입 결정;
    타입변환 연산자 삽입;
    // 단위 치환문 추출
    단위치환문의 집합 S를 구함;
    // 단위치환문의 연산 우선 순위 결정
    치환문 집합에서 topological sort에 의한 우
    선 순위 결정;
end
```

(알고리즘 1) 구문분석기 알고리즘

### 3.5 사건 계산기

사건 계산기에서는 구문 분석기에서 얻어진 치환문의 우선 순위에 따라 실제 가시화층의 뷰 입력값으로 들어갈 수 있도록 계산을 행한다. 사건 필터로부터 입력사건변수의 갱신이 이루어지면 입력된 변수에 의존적인 단위 치환문을 선택하여 우선 순위 큐에 넣는다. 우선 순위 큐에 들어간 치환문중 우선 순위가 가장 높은 치환문을 선택하여 치환문을 계산한다. 계산이 끝나면 다음 우선 순위의 치환문을 수행하는데, 만약 좌측 변수값 즉, 단위 치환문이 갱신되면 좌측변수의 값에 의존적인 치환문을 우선 순위 큐에 삽입한다. 이와 같은 과정을 반복하여 마지막으로 최종 계산된 성능 데이터가 뷰의 입력값으로 들어가게 된다.

```
Algorithm for 사건 계산기
Input : 사건 데이터  // 최근에 갱신된 입력변수
begin
  for 사건 데이터
    구문 큐에 입력된 사건변수에 의존하는 단위치
    환문 삽입;
  while 구문 큐가 비어 있지 않는 동안
    구문 큐에서 가장 높은 우선 순위 단위치환문
    제거;
    가장 높은 우선 순위 단위치환문 실행
      if 왼쪽 변수 갱신
        if 왼쪽 변수 = 출력사건변수
          뷰 입력 호출;
        else
          add 구문 큐에 왼쪽 변수에 의존하는 단
          위치환문 삽입
      endif
    endif
end
```

(알고리즘 2) 사건 계산기 알고리즘

## 4. 사건 표현식을 이용한 성능 분석

병렬 프로그램의 성능 분석은 원시 프로그램에 추적 라이브러리를 삽입하여 추적된 파일에서 성능 분석에 관심 있는 데이터를 추출하고, 추출된 성능 데이터를 사건 표현식을 이용하여 기술한다. 기술된 사건 표현식은 사용자가 정의한 사건 표현식에 의해 사건 변수에 저장된 값을 계산한다. 성능 분석층에서 필터링된 성능 데이터는 가시화층에서 뷰를 통하여 실제 분석하고자 하는 정보를 쉽게 관찰할 수 있도록 한다.

다음은 메시지 전달 프로그램에서 큐를 통하여 메시지를 주고받는 프로그램을 사건 추적하는 과정이다. (그림 9)는 원시 프로그램에서 메시지가 송·수신될 때 메시지 큐를 조사하기 위해 추적 함수를 삽입하였다. 추적 함수는 프로그램 수행시 사건에 대한 추적 데이터를 생성한다.

```
sendmsg(q[i], "hello");
trace("SEND", i); /* 인스트루멘테이션 코드 */
                .
                .
recvmsg(q[i], &msg);
trace("RECV", i); /* 인스트루멘테이션 코드 */
```

(그림 9) 인스트루멘테이션된 프로그램

(그림 10)은 인스트루멘테이션을 통하여 추출된 사건 추적 데이터이다. 삽입된 추적 함수에 따라 메시지가 발생하는 시점에서 발생하는 사건 데이터는 메시지가 나오는 시간의 타임 스탬프와 메시지가 나오는 큐에 대한 정보를 가지고 있다. SEND는 메시지가 보내어질 때 발생하는 시간과 메시지가 도착해야 하는 큐에 대한 정보를 나타내고, RECV는 메시지를 받았을 때의 시간과 큐의 번호를 기록한다.

```
SEND {
   long time; /* 타임스탬프 */
   int q;     /* 큐번호 */      }

RECV {
   long time; /* 타임스탬프 */
   int q;     /* 큐번호 */      }
              SEND  1000   1
              SEND  1005   2
              SEND  1010   1
                    .
                    .
                    .
              RECV  1035   6
              SEND  1040   7
              RECV  1045   3
```

(그림 10) 사건 추적 데이터

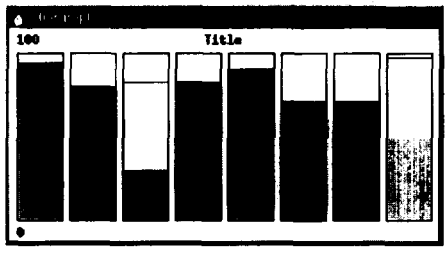
(그림 11)은 성능 분석을 위해 사건 표현식을 작성한 예이다. SEND와 RECV는 사건 추적 파일에 기록된 입력사건변수이고, Scnt와 Rcnt는 임시사건변수로서 사

건표현식에 의해 계산된 값을 임시로 기록하는 변수이다. Qlen는 출력사건변수로서 가시화 층에 입력값으로 사용된다. 사건 표현식을 이용하여 메시지가 발생했을 때 각각 메시지가 발생한 큐의 Scnt와 Rcnt를 계산하여 그 차를 구함으로써 매 시간에 대해 큐에 현재 남아있는 메시지 큐의 양에 대한 정보를 구할 수 있다.

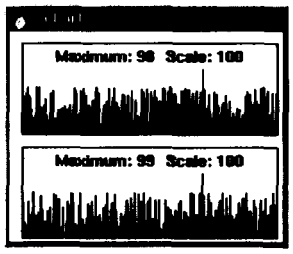
```
int Scnt[8]
int Rcnt[8]
int Qlen[8]

Scnt[SEND.q] += 1 if (SEND);  .....(1)
Rcnt[RECV.q] += 1 if (RECV);  .....(2)
Qlen = Scnt - Rcnt;           .....(3)
```

(그림 11) 사건 표현식

(그림 12)는 사건 표현식에 의해 필터링된 데이터를 가시화 뷰를 통해 분석한 것이다. 막대 그래프와 차트 그래프는 현재 큐에 남아 있는 메시지의 양을 나타낸다. 막대 그래프는 8개 노드의 메시지 큐에 남아 있는 각각 메시지의 양을 보여준다. 1번과 5번 노드는 남아 있는 메시지의 양이 많음으로서 처리해야할 작업이 많음을 알려주고 있고, 3번과 8번 노드는 남아 있는 메시지의 양이 작음을 알 수 있다. 챠트 그래프는 매 시간 메시지 양의 증감에 따라 막대 그래프의 길이가 달라짐을 볼 수 있는데 이를 통하여 프로그램의 진행 상태에 따라 시간 흐름에 따른 메시지 양을 비교 분석해 볼 수 있다. 이와 같이 사건 표현식을 이용하면 프로그래머가 성능 분석층에서 분석하고자 하는 정보를 원하는 형태로 가공할 수 있다. 또한, 프로그래머는 가공된 정보를 이용하여 가시화층에서 병렬 프로그램이 효율적으로 짜여져 있는지 성능에 개선점이 있는지 쉽게 판단하고 수정할 수 있다.




(a)




(b)

(그림 12) 성능 분석을 위한 가시화

## 5. 결 론

병렬 프로그램의 성능 분석을 위한 도구는 병렬 컴퓨터의 복잡성과 병렬 프로그래밍의 어려움으로 인해 많은 연구들이 진행되어져 왔으나, 특정 하드웨어에 의존적이거나 성능 분석 방법이 다양하지 못하다는 단점을 가지고 있었다.

본 논문에서는 다양한 성능 분석 방법을 제공하기 위해 사건 표현식을 설계하고, 이를 성능 감시 도구의 성능 분석층에 적용하였다. 제안한 성능 분석층은 사건 추적층, 가시화층과 독립적으로 개발하여 다른 층과의 인터페이스만 정의되면 다른 도구들과의 응용이 가능하다. 또한, 사건 표현식에서 제공하는 연산자와 함수를 이용하여 사용자가 직접 성능 분석 과정을 기술함으로써 사용자의 입장에서 다양한 분석이 가능하다. 사건 표현식에 의한 결과 값은 가시화층에 입력값으로 전달되어 사용자의 요구에 맞는 성능 분석을 할 수 있다.

사용자는 사건 표현식을 사용하여 관찰하고자 하는 사건에 대한 기술을 쉽게 할 수 있으며, 사건 데이터 필터링, 처리 흐름, 추적 파일 양식의 내부 구조 등 내부 처리 과정을 사용자에게 숨김으로써, 사용자가 간단한 사건 표현식으로 프로그램의 성능 분석이 가능하도록 하고 있다. 또한 사용자가 정의한 사건 변수의 이름을 그대로 사용하고, 사건 변수에 대한 오버로딩 기능을 제공하여 간단하고 다양한 분석환경을 제공해 준다. 그리고, 사건 표현식은 일반 프로그램의 수식과 유사하여 사용이 간편하며, 제공되는 연산자를 통하여 특정 사건의 검출에 쉽게 이용될 수 있다.

앞으로의 연구과제는 사건 추적층의 여러 형태의 사건 추적 파일을 접근 할 수 있는 인터페이스 개발과 성능 분석층에서 제공하는 통계 함수를 다양화하고,

사건 표현식에 의한 분석 결과를 쉽게 나타내줄 수 있는 확장성 있는 가시화층의 개발이 필요하다.

## 참 고 문 헌


[1] M. T. Heath and J. A. Etheridge, "Visualizing the performance of parallel programs," IEEE Software Vol.8, No.5, pp.29-39, Sep., 1991.

[2] Ruth A. Aydt, "An Informal Guide to Using Pablo," Dept. of Computer Science, Univ. of Illinois, May 12, 1992.

[3] Bjarne Stroustrup, "The C++ Programing Language Second Edition," ADDISON-WESLEY PUBLISHING COMPANY, 1993.

[4] Doreen Y. Cheng, "A Survey of Parallel Programming Language and Tools," Report RND-93-005, Ames Research Center, NASA, March 1993.

[5] Barton P. Miller, "What to Draw? When to Draw? An Essay on Parallel Program Visualization," Journal of Parallel And Distributed Computing 18, pp.265-269, 1993.

[6] R. Hofmann, R. Klar, B. Mohr A. Quick, and M. Siegle "Distributed Performance Monitoring: Methods, Tools, and Applications," IEEE Transactions On Parallel and Distributes Systems, Vol.5, No.6 pp.585-597, JUNE 1994.

[7] Lawrence A. Crowl, "How to Measure, Present, and Compare Parallel Performance," IEEE, pp.9-25, Spring, 1994.

[8] Tom Hudson, "Panorama's Logging Subsystem," Dept. of Computer Science and Engineering, Univ. of California, 1994.

[9] Elizabeth E. Frank, Ruth A. Aydt, "The Pablo Performance Visualization System Functional Specification," Dept. of Computer Science, Univ. of Illinois, February 3, 1995.

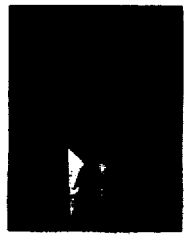
[10] Jerry Yan, Sekhar Sarukkai and Pankaj Mehra, "Performance Measurement, Visualization and Modeling of Parallel and Distributed Programs using the AIMS Toolkit," Software-Practice and Experience, Vol.25(4), pp.429-461, April, 1995.

[11] 마대성, 유진호, 이상준, 김정선, 김정훈, 지동해, 김병기, "병렬 프로그램의 성능 분석을 위한 사건 표현식 설계", 전자공학회논문집, 제18권, 제2호, pp. 429-432, 1995.

[12] E. Kraemer and J. T. Stasco, "The Visualization of Parallel System : An Overview," Journal of Parallel and Distributed Computing, Vol.18, pp.105-117, 1993.

[13] Ruth A. Aydt, "The Pablo Self-Defining Data Format," Dept. of Computer Science, Univ. of Illinois, March, 1993.

[14] Daniel A. Reed, Ruth A. Aydt, Luiz DeRose, Celso L. Mendes, Randy L. Ribler, Eric Shaffer, Huseyin Simitci, Jeffrey S. Vetter, Daniel R. Wells, Shannon Whitmore, and Ying Zhang, "Performance Analysis of Parallel Systems : Approaches and Open Problems," Joint Symposium on Parallel Processing(JSPP), pp.239-256, 1998.


**김 병 기**

e-mail : bgkim@chonnam.chonnam.ac.kr

1978년 전남대학교 수학과 졸업(이학사)

1980년 전남대학교 대학원 수학과 졸업(이학석사)

1996년 전북대학교 대학원 수학과 박사과정 수료

1981년~현재 전남대학교 컴퓨터정보학부 교수

1994년~현재 한국정보처리학회 이사

관심분야 : 소프트웨어공학, 병렬처리, 소프트웨어 에이전트, 객체지향 시스템



**마 대 성**

e-mail : dsma@mudeung.kwangju-e.ac.kr

1994년 호남대학교 전산통계학과 졸업(이학사)

1996년 전남대학교 대학원 전산통계학과 졸업(이학석사)

1998년 전남대학교 대학원 전산통계학과 박사과정 수료

1997년~현재 광주교육대학교 전산교육과 조교

관심분야 : 병렬처리, 소프트웨어공학, 객체지향시스템