사용자 정의 성능 분석을 지원하는 성능 감시 도구의 설계 및 구현

마 대 성* · 김 병 기**
광주교육대학교 전산교육과 · 전남대학교 전산학과

요 약

본 논문에서는 RDL/PAL 인터페이스와 사건표현식을 지원하는 성능 감시 도구를 설계하였다. RDL/PAL은 성능 감시 도구의 사건 추적층과 성능 분석층간의 인터페이스를 제공하고, 임의의 추적 양식에 접근하여 사건 데이터를 추출해낼 수 있다. 또한, 사건표현식은 일반적인 프로그래밍 언어와 유사한 형태로 구성되어 프로그래머가 원하는 성능 분석 결과를 쉽게 얻을 수 있다. 프로그래머는 본 논문에서 구현한 성능 감시 도구를 이용하여 추적 양식에 관계없이 다양한 성능 분석을 할 수 있다는 장점을 가지고 있다.

Design and Implementation of Performance Analysis Tool For User-Defined Performance Analysis

Daisung Ma* · Byungki Kim**
Kwangju National University of Education, Dept. of Computer Science Education*
Chonnam National University, Dept. of Computer Science**

ABSTRACT

This paper designs RDL/PAL interface and performance monitoring tool with providing the Event Expressions. RDL/PAL suggests interface of instrumentation layer and performance analysis layer so that it can extract event data from various trace formats. Also, the Event Expressions is similar to the normal program language, which make possible to obtain result of performance analysis. The programmer can analyze performance regardless of trace format by using the performance monitoring tool which has so much merits.

1. 서론

병렬 프로그램은 수행되는 시스템의 환경에 의존적이고, 구성요소간의 상호교류로 인해 효율적인 프로그래밍을 하기가 매우 어렵다. 이로 인해 프로그래머는 기대에 맞지 않는 성능과 원하지 않은 결과를 얻기도 한다. 이와 같은 문제점을 해결하기 위해 병렬 프로그램의 복잡한 동작 특성들을 시각화하여 보여 주는 성능 감시 도구들이 개발되어지고 있다[1,2,3,4].

그러나, 기존 성능 감시 도구들의 가장 큰 단점은 첫째, 제한된 시스템에서 동작하도록 구현되었다는 것이다. 성능 감시 도구들이 다른 시스템으로 이식될지라도 기 구현된 병렬시스템의 특성에 맞게 개발되었기 때문에 이식된 시스템에서는 프로그래머가 얻고자 하는 성능 분석 정보를 제공하지 못하는 경우가 발생한다.

둘째, 성능 감시 도구의 종류가 시스템마다 다르고, 분석 방법이 서로 상이하다는 점이다. 프로그래머는 시스템이 바뀔 때마다 새로운 분석 도구를 익혀야 한다.

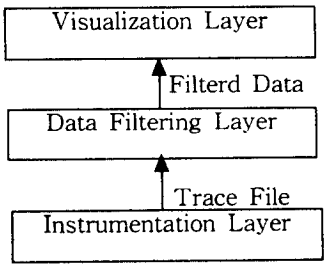
셋째, 기존의 병렬 프로그램 성능 감시 도구들은 서로 다른 물리적 추적 양식(trace format)과, 그 추적 양식에 맞는 접근 방법들을 제공하고 있기 때문에 주어진 추적 양식의 추적 데이터를 서로 다른 성능 분석 도구에서 분석 할 수 없다는 단점을 가지고 있다.

일반적으로 이러한 문제점은 주어진 추적 양식의 데이터를 특정한 추적 양식으로 변환시키는 변환기를 사용하여 해결하고 있다[4]. 그러나, 변환기를 사용하는 방법에는 몇가지 문제가 있다. 모든 추적 양식에 대한 변환기를 두어야 하고 각 추적 양식이 새로운 요구사항에 맞게 변화되는 경우 해당되는 변환기를 그때마다 바꾸어야 한다는 단점을 가지고 있다.

본 논문에서는 이와 같은 문제점을 해결하기 위해 사용자 정의 성능 분석을 지원하는 성능 감시 도구를 설계하고 구현하였다. 제안하는 성능 감시 도구는 사건 추적층과 성능 분석층 사이에 RDL/PAL 인터페이스[5]를 이용한다. RDL/PAL 인터페이스는 사건 추적 양식에 관계없이 성능 분석층에서 추적데이터를 접근할 수 있는 방법을 제공하여 다른 성능 분석 도구에서 추적된 데이터를 이용할 수 있다. 성능 분석층에서는 사용자 정의 분석을 지원할 수 있도록 사건 표현식[6]을 제공한다. 사건 표현식은 일반적인 프로그래밍 언어의 표현식[7]과 유사하여 프로그래머가 추적 데이터를 이용하여 쉽게 성능 분석이 가능하도록 하였다. 가시화층에서는 확장성 있는 일반화된 뷰[8]를 제공한다. 일반화된 뷰는 병렬 컴퓨터의 특성에 좌우되지 않고 범용적인 분석 결과를 제공한다는 장점을 가지고 있다. 본 논문의 구성은 다음과 같다. 2장은 일반적인 성능감시도구에 대해 알아보고 3장에서는 RDL/PAL에 대해 기술한다. 4장에서는 성능 분석층에서 사건 표현식에 대해 기술하고 5장에서는 성능 감시 도구의 구현에 대해 기술하였다. 6장에서는 결론에 대해 언급하였다.

2. 일반적인 성능 감시 도구의 고찰

성능 감시 도구는 프로그램의 성능 분석에 필요한 사건을 수집하는 사건 추적층, 사건 추적층에서 수집된 데이터로부터 성능 분석 데이터를 추출하는 성능 분석층, 필터링된 데이터를 이용하여 이해하기 쉬운 형태로 나타내어 주는 가시화층으로 구성되었다. 성능 감시 도구는 현재 이와 같은 계층적 구조를 기초로한 설계가 이루어지고 있다. 계층적 층구조는 인접한 층간의 접속부(interface)가 잘 정의되면 각 층은 독립적이며 효율적으로 개발 될 수 있고, 임의의 병렬 시스템에도 잘 적용될 수 있어, 개방성(openess)과 이식성(portability) 그리고, 확장성(extensibility)을 갖는다. 계층적 층구조를 갖는 성능 감시 도구는 <그림 1>과 같다.



<그림 1> 일반적인 성능 감시 도구의 구조

사건 추적층(Instrumentation Layer)은 어떠한 사건을 정의할 것인가, 어떤 종류의 사건 데이터를 수집할 것인가, 목적 시스템으로부터 병렬프로그램의 성능에 영향을 미치는 어떤 환경을 어떻게 감시할 것인가 등에 따라 사건을 수집하는 계층이다. 사건 추적층에서 수집되는 사건 데이터는 각각의 사건에 대한 식별자와 속성을 갖고 있다. 사건의 속성으로는 타임스탬프(timestamp)와 사건에 관련된 특정 데이터들을 포함한다. 사건을 수집하기 위해 성능 감시 도구는 인스트루멘테이션(instrumentation)을 행한다.

인스트루멘테이션에 의해 수집된 사건 데이터는 대규모 프로그램에서는 사건의 수집 범위에 따라 데이터의 양이 방대해진다. 프로그램의 성능 분석을 위해 수집된 방대한 양의 사건 데이터는 성능 분석에 관계없는 경우가 있고, 의미 있는 데이터라 할지라도 프로그래머에게 관심 있는 시간의 범위에 속하지 않는 경우에는 무시해야 한다.

성능 분석층(Data Filtering Layer)은 수집된 사건 데이터를 이용하여 프로그래머가 관심 있는 분석을 행할 수 있도록 필터링(filtering)과정을 수행한다. 필터링과정은 수집된 사건 데이터를 프로그래머에게 의미 있는 데이터로 바꾸어 주기 위한 과정으로 잘 형식화된 정보(well-formatted)를 제공한다. 필터링을 위한 방법으로는 여러 가지 방법이 있으나 크게 두 가지 방법으로 구분할 수 있다.

첫째, 성능 감시 도구 내부에서 제공되는 필터링 모듈에 따라 제공되는 방법 즉, 분석하고자 하는 필터링과정을 성능 감시 도구가 내부에 정의하여 성능 분석 데이터를 추출해 낸다.

둘째, 일반 디버거에서처럼 사용자가 분석하고자 하는 정보를 필터링 모듈에 기술하면, 기술된 필터링 규칙에 의해 성능 데이터를 생성해낸다. 사용자의 정의에 의한 필터링 과정은 사용자가 분석하고자 하는 데이터를 추출해 낼 수 있다는 장점이 있다.

ParaGraph[9]과 같은 경우는 인스트루멘테이션 단계에서 수집된 사건 데이터를 시스템 내에서 제공하는 필터링 모듈에 의해 정보를 제공한다. 이 방식의 장점은 분석 과정이 쉽다는 장점이 있는 반면에 필터링 모듈에 정의되지 않은 성능 정보는 사용자가 볼 수 없다는 단점이 있다.

Pablo[3,10]는 사용자가 타원과 선을 이용하여 필터링하는 과정을 그래픽으로 표현하고 있다. 이러한 방식은 사용자의 이해를 쉽게 돕는다는 장점은 있지만 성능 분석을 위해 참조되어야할 많은 사건이 있는 경우 한 화면안에 모두 표현하기가 어렵다는 단점이 있다.

가시화층(Visualization Layer)은 성능 분석을 위해 실제적이고 효과적으로 분석에 도움을 줄 수 있는 가시화 뷰(view)를 제공한다. 가시화 뷰는 분석의 용도에 따른 특성화된 뷰와 추상화된 뷰를 제공한다.

특성화된 뷰는 분석하고자 하는 시스템과 프로그램의 모델에 따른 뷰로서 일반적으로 커뮤니케이션, 프로세서 부하량, 프로세스 생성량, 부하균형, 프로세서 아이들상태 등을 보여준다. 메시지 기반형 모델에는 메시지의 특성에 맞는 뷰 즉, 각 프로세스의 메시지의 양, 메시지의 빈도, 메시지의 송·수신 상태, 메시지의 lock, unlock상태 등을 나타내주는 뷰 등을 보여준다[9]. 쓰레드 뷰는 쓰레드의 생성과 파괴 등을 보여준다.

추상화된 뷰는 뷰에 제공되는 정보가 무엇이든지 관계없이 성능 데이터의 상태를 보여주는 막대그래프(Bar graph), 행렬 그래프(Matrix graph), 키비어트 그래프(Kiviat graph), 간트 그래프(Gant graph) 등이 있다. 사용자는 분석 하고자 하는 목표에 따라 성능을 측정하기에 적당한 추상화된 뷰를 선택하여 성능을 측정할 수 있다[3].

3. RDL/PAL

RDL/PAL 인터페이스[5]는 성능 감시 도구에 따라 서로 다른 추적 양식을 접근할 수 있는 인터페이스를 제공해준다. 본 논문에서 제안하는 성능 감시 도구의 사건 추적층에서는 RDL/PAL을 이용하여 추적 파일에 접근한다.

3.1 RDL

RDL(Record Description Language)은 임의의 추적 양식을 기술하기 위한 언어로서 추적 설명자라고도 한다. 필드설명자(Field Descriptor)는 레코드 설

명자의 각 필드의 타입과 이름을 묘사한다. 레코드 설명자는 (Record Descriptor)는 논리적 레코드 타입의 이름과 구조, 그리고 표현을 나타낸다. 하나의 레코드 설명자는 하나 이상의 필드 설명자들로 구성된다. 바인드 설명자(Bind Descriptor)는 레코드 실체의 구조와 표현에 관한 정보를 가지고 있는 레코드 설명자를 식별하는데 필요한 바인딩 정보를 명세한다. RDL 추적 양식은 물리적 양식들과는 달리 고정되어 있지 않고 SDDF[3]와 같이 자기 서술 또는 자기 정의 능력을 가지고 있기 때문에 데이터의 구조와 표현에 관한 정보를 유연하게 포함할 수 있다. <그림 2>은 ParaGraph에서 제공하는 PICL 양식을 RDL을 이용하여 기술한 예이다.

```
DESCRIPTION "PICL Format" IS
BEGIN

RECORD "Trace Start" IS //Record Descriptor
BEGIN
 int    "Trace Start"; //Field Descriptor
 int    "Seconds";
 int    "MicroSeconds";
 int    "Node";
 int    "Event";
 int    "Computation Stats";
 int    "Communication Stats";
END;

RECORD "Open" IS
BEGIN
 int    "Open";
 int    "Seconds";
 int    "MicroSeconds";
 int    "Node";
 int    "Processors Allocated";
END;

BIND IS  // Bind Descriptor
        KEYFIELD INDEX IS (0, NONE);
BEGIN
   <"1"> IS  "Trace Start";
   <"2"> IS  "Open";
END;

END.
```

<그림 2> RDL 기술 예

3.2 PAL

PAL(Problem oriented trace Access Library)[5]은 추적 화일에 있는 추적 데이터를 엑세스할 수 있도록 하여 주는 인터페이스 함수들로 구성된 라이브러리이다. PAL은 미리 생성한 이진 추적 설명자를 파일로부터 읽어 들이거나, 실행시에 RDLC 컴파일러를 실행시켜 RDL 소스화일을 직접 처리하여 읽어 들일수 있다.

새로운 추적 양식을 처리하기 위해서는 RDL을 이용하여 주어진 추적 양식을 기술하는 RDL 소스화일을 생성하다. RDL 소스화일은 RDLC 컴파일러를 이용하여 이진 추적 설명자 파일을 생성한다. 마지막으로 성능 감시 도구 실행 시에 PAL을 통하여 이진 추적화일을 접근하여 처리할 수 있도록 하였다.

<그림 3>은 PAL을 통하여 추적 파일을 접근하는 예를 보여준다.

```
#include "Rdl.h"
#include "Dictionary.h"
#include "SreamPipe.h"
#include "Reader.h"

main()
{
 Dictionary      *RD;
 StreamPipe      *inFile;
 Reader          *reader;

 RD=new Dictionary;
 RD->build("RDL.def");

 inFile = new SreamPipe("trace.trf")
 reader = new Reader (inFile, RD);

 do_something (reader);
 inFile->close();
}
```

<그림 3> PAL을 이용한 추적화일 접근방법

4. 사용자 정의 성능 분석

본 장에서는 성능 감시 도구의 성능 분석층에서 사용자가 정의한 사건 표현식에 대해 기술한다.

4.1 사건 표현식

사건 표현식은 사건 추적 파일에 기록되어 있는 사건을 분석하는데 이용된다. 사건 표현식의 구성은 사건 변수, 상수, 연산자, 함수들로 구성한 표현식이며 일반적인 프로그래밍 언어에서의 표현식과 유사하다. 프로그래머는 사건 추적 파일내의 사건들을 조합하여 사건 데이터의 통계나 프로그램의 성능에 영향을 끼치는 분석 결과를 얻는다. 사건표현식에서의 계산은 사건 표현식에 포함된 사건 변수의 값이 변할때마다 다시 계산되어 새로운 값으로 계산된다. <그림 4>은 특정 노드에서 메시지의 송·수신에 의한 큐의 길이를 구하는 사건표현식의 예이다.

```
int SCNT[7]
int RCNT[7]
int QLEN[7]

SCNT[SEND.q] += 1 if (SEND)
RCNT[RECV.q] += 1 if (RECV)
QLEN = SCNT - RCNT;
```

<그림 4> 사건 표현식의 예

4.2 사건 표현식의 문법

사건 표현식의 문법은 일반 프로그래밍 언어와 유사하다. <그림 5>과 같이 사건표현식의 문법을 정의하였다.

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

<그림 5> 사건 표현식의 문법

4.3 사건 변수

프로그래머는 분석하고자 하는 사건에 대해 사건 추적화일 내의 사건 정의 이름을 사건 표현식에 기술한다. 사건변수는 기능에 따라 입력사건변수, 출력사건변수, 임시사건변수등 세 가지 종류로 분류된다.

입력사건변수는 사건 추적 파일 내에 정의된 사건 이름이다. 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.

4.4 연산자 및 함수

사건 표현식은 사건변수, 상수, 연산자와 함수들을 사용하여 기술한다. 사건표현식에서 제공하는 연산자는 산술연산자, 논리연산자, 관계연산자, 비트연산자, 치환연산자 등의 C 언어의 표현식에서 사용되는 대부분의 연산자와 함께 사건에 대한 필터 연산자를 추가하여 지원한다.

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. 필터연산자를 이용하여 조건부 사건 필터링이

가능하다.

또한, 사건 표현식에서는 성능 분석에 필요한 각종 통계 데이터들과 성능 분석 데이터들을 다른 도구들의 도움 없이 성능 분석과정에서 쉽게 추출할 수 있도록 함수들을 제공한다.

<표 1> 사건 표현식의 연산자와 통계함수

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

4.5 연산자/함수 오버로딩

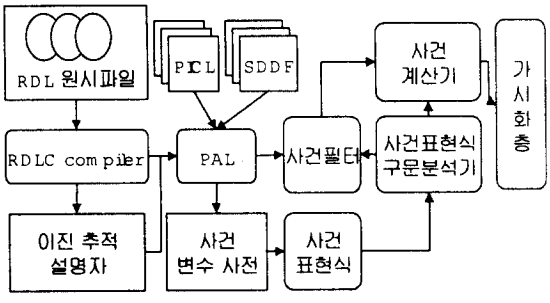
C++언어에서 지원되는 연산자와 함수에 대한 오버로딩(overloading)기능[7]을 사건표현식에서도 제공한다. 예를 들면 '+' 연산자에는 정수, 실수, 벡터, 배열, 문자열 데이터를 피연산자로서 허용한다. sum( ) 는 매개 변수로 정수, 실수, 벡터, 배열을 허용한다. 각 연산자와 함수의 리턴값의 데이터형식은 피연산자와 매개 변수의 데이터 형식에 따라 결정된다. 본 논문의 사건 표현식에서는 벡터와 배열에 대한 연산자/함수 오버로딩 기능을 별도로 추가하였다.

또한, 사건 표현식에서 어떤 연산자와 피연산자들 사이의 데이터형이 같지 않거나 어떤 함수의 매개 변수로서 요구되는 데이터의 형이 실제 매개 변수의 형과 일치하지 않을 경우, 구문분석단계에서 데이터형 변환 연산자가 자동적으로 삽입되어 강제적으로 데이터형을 변환하도록 하여 프로그래머가 사건 표현식을 쉽게 기술할 수 있도록 하였다.

5. 성능 감시 도구 구현

사용자 정의 성능 분석을 지원하는 성능 감시 도구의 전체 구조는 <그림 6>과 같다. 성능 감시 도구는 3개 층으로 이루어진다. 사건 추적층에서는 임의의 사건 추적 양식을 접근하여 사건 데이터를 다음 성능 분석층에 보내주는 역할을 한다. 임의 사건 추적 양식은 RDL에 의해 기술된다. RDL에 의해 기술된 추적 설명자는 RDLC 컴파일러에 의해 이진 추적 설명자로 변환된다.

성능 분석층에서는 PAL을 통하여 추적 양식에 접근한다. PAL은 이진 추적 설명자에서 정의한 레코드와 필드값을 이용하여 추적 화일에 접근을 가능하게 한다. 프로그래머는 사건표현식을 작성하여 추적 파일에 기록되어 있는 사건들을 분석할 수 있다. 사건 표현식 구문 분석기는 프로그래머가 작성한 사건 표현식을 분석하여 우선 순위에 의한 단위 치환문들을 생성한다. 이 단계에서는 어휘분석과정을 거쳐 사건변수 토큰, 예약어 토큰, 연산자 토큰들로 구분된다. 추적 파일의 사건은 해당행위가 이루어질 때만 발생하기 때문에 동기적으로 발생하지는 않는다. 그래서, 추적 파일내의 각 사건들은 사건이 발생한 시간 정보를 가지고 있다. 사건 계산기에서는 해당 사건 데이터의 의존관계에 따라 우선 순위 큐에 들어있는 단위 치환문들을 계산한다. 마지막 단위치환문이 계산이 되면 가시화층의 뷰의 입력으로 들어간다. 가시화층에서는 각 뷰에 대한 동기화를 위해 성능 분석층과 뷰 사이의 이벤트 핸들러가 뷰를 제어한다. 또한 각 뷰에는 Call-Back 기능을 제공하여 사용자가 애니메이션되는 뷰의 관심 있는 데이터값을 보고자 할 때 마우스를 클릭하면 마우스 위치의 현재 값을 디스플레이 한다.



<그림 6> 성능 감시 도구의 전체 구조도

6. 결론

성능 감시 도구는 병렬 컴퓨터의 시스템 구조에 종속적이어서 범용의 성능 감시 도구를 개발하는 것은 매우 어렵다. 이와 같은 이유로 병렬 프로그램을 분석하기 위한 도구들은 시스템 종류마다 달라 프로그래머는 매번 새로운 성능 감시 도구의 기능에 익숙해져야 한다는 문제점을 지니고 있다.

사용자 정의 성능 분석을 지원하는 성능 감시 도구는 이미 개발되어 사용중인 각 병렬 컴퓨터의 성능 감시 도구에서 추출되는 임의의 추적 파일을 억세스하여 분석할 수 있도록 하였다. 또한, 임의의 추적 양식내의 추적 데이터의 내용과 수준은 추적 대상 시스템이나 프로그램 모델, 추적의 정도에 따라 달라지게 된다. 서로 다른 추적 데이터의 내용을 성능 감시 도구가 포용하여 분석할 수 있는 방법을 제공하기 위해서는 좀 더 유연한 성능 분석을 위한 방법이 요구되었다.

본 논문에서는 사건 추적층에서는 RDL/PAL 인터페이스를 이용하여 어떤 형태의 사건 추적 양식도 수용할 수 있도록 하였고, 성능 분석층에서는 사용자 정의 분석을 지원하기 위해 사건표현식을 이용하여 프로그래머의 의도대로 분석이 가능하도록 설계하였다. 사건 표현식을 제공하는 성능 감시 도구는 프로그래머가 일반 언어와 유사한 표현식으로 사건에 대한 분석을 할 수 있기 때문에 프로그램의 특정 데이터 상태, 프로그램의 제어 상태, 사건의 성능 통계분석, 특정 사건 검출 등 다양한 형태의 성능 분석을 제공한다.

참고문헌

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

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

[3] D. A. Reed, R. A. Aydt, T. M. Madhyastha, R, Noe, K. Shields and B. Schwartz, "An Overview of the Pablo Performance Analysis Environment," University of Illinois Board of Trustees, November, 1992.

[4] 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.

[5] 김정선, 김정훈, 지동해, "사건기반 성능 가시화 시스템을 위한 문제중심의 추적 인터페이스", 한국정보과학회 추계학술발표논문집, 제22권, 제2호, 1995.

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

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

[8] 마대성, 유진호, 김병기, "성능 감시기의 가시화층을 위한 일반화된 뷰의 설계", 한국 정보과학회 학술발표논문집, 제25권, 제2호, pp756-758, 1998.

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

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