병렬 프로그램 성능가시화를 위한 확장성 있는 프레임워크 설계 및 구현

(Design and Implementation of a Scalable Framework for Parallel Program Performance Visualization)

문 상 수 † 문 영 식 †† 김 정 선 ††
(Sangsu Moon) (Youngshik Moon) (Jungsun Kim)

요 약 본 논문에서는 최적의 성능을 갖는 병렬프로그램 개발을 위한 도구로 이식성, 확장성, 효율성을 고려한 성능가시화 프레임워크의 설계 및 구현을 제시한다. 본 프레임워크는 독립적으로 개발 가능한 인스트루멘테이션 층, 인터페이스 층 그리고 가시화 층으로 이루어진 계층구조를 갖도록 설계되었다. 인스트루멘테이션 층은 사건(event) 포획을 위한 라이브러리인 ECL(Event Capture Library)로 구성되며, 인터페이스 층은 인스트루멘테이션 층과 가시화 층 간에 문제중심의 인터페이스를 제공하기 위해 개발된 EDL/JPAL(Event Description Language/Java Problem-oriented trace Access Library)로 구성되었다. 또한 가시화 층은 뷰와 필터의 추가, 수정 및 custom 뷰 그룹의 구성이 용이하도록 plug and play 스타일로 설계되었다. 이렇게 구현된 성능 가시화 프레임워크는 독립된 도구로서 뿐만 아니라 프로그래밍, 디버깅 그리고 성능분석이 통합된 병렬프로그램 개발환경의 핵심도구로 사용될 수 있다.

Abstract In this paper, we propose the design and implementation of a portable, extensible, and efficient performance visualization framework for high performance parallel program development. The framework adopts a layered architecture:consists of three independent layers instrumentation layer, trace interface layer and visualization layer. The instrumentation layer was constructed as an ECL which captures generated events, and the EDL/JPAL constitutes the trace interface layer to provide problem-oriented interfaces between visualization layer and instrumentation layer. Finally, the visualization layer was designed as plug-and-play style for easy elimination, addition and composition of various filters, views and view groups. The proposed performance visualization framework is expected to be used as an independent performance debugging and analysis tool and as a core component in an integrated parallel programming environment.

1. 서 론

초창기의 병렬 컴퓨터들은 수십 개의 프로세서들이 메모리를 공유하는 공유메모리 (shared memory) 방식을 취했으나, VLSI 및 마이크로프로세서의 기술이 비약적으로 발전함에 따라 최근 들어 수백 또는 수천 개의 프로세서/메모리들로 구성된 분산 메모리 형태의 대규모 병렬 컴퓨터(Massively Parallel Processor (MPP))가 등장하기 시작하였다. 이러한 대규모 병렬 컴퓨터는 단일 프로세서로 구성된 기존의 슈퍼컴퓨터에 비하여 가격 대 성능 면에서 훨씬 우수하다. 엄청난 계산량을 필요로 함으로써 이전까지 슈퍼컴퓨터를 사용해서도 불가능했던 자연과학 및 공학분야의 소위 Grand challenge problem들 --- 실시간 영상 정보처리, 3D Computational Fluid Dynamics, 기후 모델링, 인공 지능, VLSI 설계 자동화 등 --- 이 대규모 병렬 컴퓨터를 통하여 해결 가능하게 되었다. 이렇게 병렬 컴퓨팅의 하드웨어의 기술은 비약적인 발전을 이루어 왔지만, 상대적으로 병렬 소프트웨어 개발 기술은 아직 국내외적으로 초기 단계에 지나지 않는다. 이처럼 병렬 소프트웨어 활성화

† 비 회 원 : (주)엠헨즈 대표이사
ssmoon@mhenz.com
†† 종신회원 : 한양대학교 전자계산학과 교수
ysmoon@cse.hanyang.ac.kr
jskim@cse.hanyang.ac.kr
논문접수 : 2000년 6월 20일
심사완료 : 2001년 3월 13일

를 가로막는 가장 큰 요인 중에 하나로 병렬 디버거, 구문 중심 에디터, 성능가시화 프레임워크 등 통합된 병렬 프로그램 support tool의 지원이 미약한데 있다.

본 논문에서는 병렬 프로그래밍 지원 소프트웨어 가운데 가장 핵심적인 성능가시화 프레임워크에 대하여 기술한다. 병렬 프로그램의 동작 특성을 파악하기 위하여 사건 기반형 방법을 채택한다면 사건의 내용이 방대할 뿐 아니라 숫자 또는 문자로 이루어졌기 때문에 성능분석이 쉽지 않다. 이러한 어려움을 해결하기 위해 인간의 탁월한 이미지 처리능력을 활용한 가시화(visualization)를 이용하는 것이 효과적이다 [1]. 이러한 가시화는 복잡한 숫자나 문자 대신에 이미지의 형태로 시각화하여 사용자에게 제공함으로써 성능분석 및 오류의 분석을 용이하게 한다 [2,3]. 성능가시화 프레임워크는 보여주고자 하는 정보에 따라 문제중심의 인터페이스로부터 읽어 들인 사건들을 필터링 하거나 서로 관련된 사건들을 그룹화 하여 좀 더 고 수준의 정보를 추출해 내는 기능이 필요하다. 일반적으로 사용자는 여러 개의 성능 뷰를 동시에 관찰하고 그들로부터 병렬시스템의 동작 특성에 관한 통찰력을 얻을 수 있다 [4,5]. 이를 위해 성능가시화 프레임워크는 성능 정보를 다양한 뷰를 통하여 제공할 수 있어야 함으로, 객체지향 설계 방법론을 최대한 활용하여 새로운 뷰의 설계 및 추가가 용이하도록 확장성 있게 설계 및 구현하였다. 이렇게 구현된 프레임워크는 backbone 아키텍춰를 제공함으로써 Plug-and Play 방식으로 동작할 수 있도록 하였으며 새로운 뷰의 수정 및 추가가 용이하도록 세 개의 계층구조로 갖도록 설계함으로써 기존 성능가시화 도구들이 안고있는 범용성의 결여, 성능 뷰의 추가 및 수정의 어려움, 사건레코드 개선 및 확장성 결여와 같은 문제들을 해결할 수 있도록 한다.

2. 확장성 있는 성능가시화 프레임워크

본 논문에서 제시하는 성능가시화 프레임워크는 기존 가시화기에서 제공하지 못했던 이식성, 확장성 그리고 범용성을 위하여 그림 1과 같이 층 구조를 갖도록 설계되었다.

인스트루멘테이션 층은 사건의 발생, 탐지 그리고 수집을 담당하며, 인터페이스 층은 인스트루멘테이션 층과 가시화 층 간의 독립성을 보존하기 위해 논리적 추적양식과 문제중심의 액세스를 제공할 수 있도록 EDL/JPAL로 구성하였으며, 가시화 층은 수집된 사건의 분석 및 프리젠테이션 기능을 담당하는 모듈로 사용의 편리성, 확장성 그리고 유용성을 갖도록 설계하였다. 또한

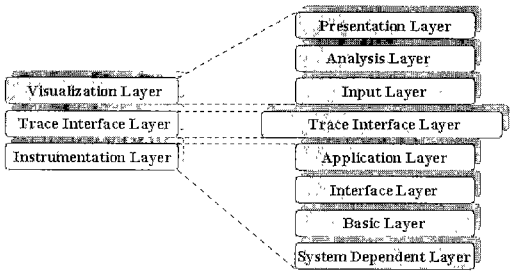

그림 1 성능가시화 프레임워크 계층구조

뷰와 필터의 추가 및 수정이 용이하도록 Software-IC로 설계 및 구현하였다.

2.1 인스트루멘테이션 층

인스트루멘테이션 층은 그림 2에서와 같이 4개의 층으로 이루어진 계층구조로 이루어져 있다.

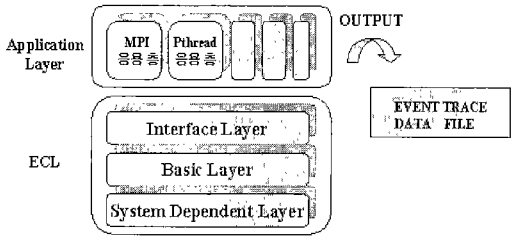

그림 2 인스트루멘테이션 층 계층구조

최상위 층은 응용 층으로 MPI[6]에 명시된 프로파일링 인터페이스에 따라 구현될 수 있도록 하였으며, 하위 세 계층은 사건포획 라이브러리인 ECL을 구성하는 계층으로 효율성, 이식성 그리고 확장성을 목표로 설계되었기 때문에 MPI 수집을 위한 인스트루멘테이션뿐만 아니라, 임의의 소프트웨어 사건 기반형 성능분석 도구를 위한 인스트루멘테이션 라이브러리로 사용될 수 있다 [7].

2.1.1 시스템 종속 층(System Dependent Layer)

ECL을 구성하는 최하위 계층으로 성능가시화 프레임워크가 다양한 플랫폼에서도 동일한 기능을 수행할 수 있는 적응성을 지니도록 해주는 목적을 따로 분리하여 모듈로 구성, 설계하였으며 플랫폼에 따라 기본 층이 요구하는 정보를 제공하며 사용자의 액세스가 불가능하다.

2.1.2 기본 층(Basic Layer)

기본 층은 ECL 계층에서 중간에 위치하며 라이브러리 층이라고도 칭한다. 기본 층은 사건 추적 정보의 저장 및 추적 파일의 생성 등, 성능가시화 프레임워크에서 내부함수를 구성하는 부분으로 기본 층에서 정의된 함수는 사용자에게 그 정보가 은닉되어 접근을 막도록 하

였다. 때문에 사용자에게 성능가시화 프레임워크 사용의 편리성을 제공해 줄 수 있으며 정보에 대한 일관성을 유지 시켜 줄 수 있는 장점을 지니게 한다.

2.1.3 인터페이스 층(Interface Layer)

인터페이스 층은 ECL에서 최상위 계층에 위치하며 실제 인스트루멘테이션을 수행하는 부분이다. 사용자가 원하는 사건에 대한 추적과 사건 추적에 대한 정보를 얻을 수 있는 사건 추적함수로 구성되며 인터페이스 층에서 제공되는 함수는 사용자가 접근 가능하다.

2.1.4 응용 층(Application Layer)

응용 층은 병렬프로그램에 대한 사건을 정의하고 사건에 대한 인스트루멘테이션을 수행하는 계층으로 본 프레임워크는 MPI 표준에 준하여 작성하였다. MPI 응용 층은 MPI 표준에서 프로파일링 함수와 동일한 의미를 지니며 그 기능 또한 같다. 즉, 인터페이스 층의 확장이라고 봐도 무방하다. 하지만 독립된 계층으로 존재하기 때문에 이식성과 확장성을 지니는 장점을 지니게 된다.

2.1.5 ECL

다음 표1은 ECL 라이브러리의 구성요소와 그 기능에 대해 기술한다.

2.2 문제중심 인터페이스 층

EDL/JPAL 인터페이스는 메타(meta) 양식의 논리적 추적양식을 지원하고, 사건레코드의 추상적인 구조와 추적 데이타에 대한 문제중심의 액세스 방법을 제공한다. 메타 추적양식은 의미(semantics)와 독립적으로 추적레코드의 논리적 구조만을 명시하며, 사건에 관한 물리적 명세를 포함하지 않으므로 임의의 추적양식의 기술을 가능하게 한다. 성능분석 시스템에서 EDL/JPAL 인터페이스를 사용하면 추적파일의 양식이 변하거나 새로운 레코드의 추가에도 영향을 받지 않으므로 대부분의 추적양식에 사용가능 하다. 그림 3은 본 성능가시화 프레임워크에서 사용하는 EDL/JPAL의 사용 예를 보여준다.

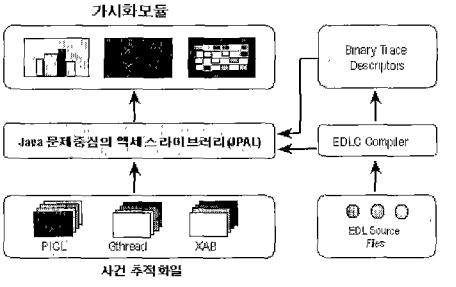

그림 3 EDL/JPAL 인터페이스

2.2.1 사건 기술 언어(EDL)

EDL은 메타 언어인 SDDF(Self-Defining Data Format)와 TDL(Trace Description Language)의 장점을 취합하여 개발하였다 [8]. EDL은 추적 파일의 레코드 구조, 표현 및 액세스 방법을 기술하는 언어로 실제 레코드 실체와 그에 해당하는 논리적 레코드 타입간의

표 1 ECL (Event Capture Library)

라이브러리	기 능
Ecl.h	인스트루멘테이션 라이브러리의 모든 최상위 수준의 자료 구조에 대한 정의와 인터페이스 함수들에 대한 선언을 포함한다.. 또한 모든 ECL 라이브러리의 사용자는 반드시 이 파일을 include 해야 한다.
EclImpl.h	인스트루멘테이션 라이브러리의 모든 최상위 수준의 자료구조에 대한 정의를 포함한다. Ecl.h에 의하여 자동으로 include 된다.
EclParam.h	인스트루멘테이션 라이브러리의 모든 *configurable* 파라미터들을 포함한다. Ecl에 의하여 자동으로 include 된다.
EclEdpend.h	인스트루멘테이션 라이브러리의 시스템 종속적인 모든 선언과 정의된 상수들을 포함한다.
EclClock.h	인스트루멘테이션 라이브러리의 요소 중 이식성 있는 클럭을 관리하는 함수들과 자료구조를 정의한다.
EclApi.c	사용자가 액세스할 수 있는 라이브러리 함수들을 포함하여 인터페이스 층을 구현한다. 여기서 정의된 함수들은 데이타의 버퍼링과 기록을 위해 라이브러리 내부 함수들을 호출한다.
EclPort.c	내부적이면서, 시스템 독립적인 함수들을 포함하며 EclClock.c와 함께 라이브러리 기본 층을 구현한다.
EclClock.c	내부적이면서, 시스템 독립적인 함수들 가운데 클럭을 관리하는 함수들을 포함하며, EclPort.c 와 함께 라이브러리 기본 층을 구현한다.
EclDepend.c	인스트루멘테이션 라이브러리의 시스템 종속적인 모든 함수들을 정의하며, 시스템 종속 층을 구현한다.
EclColock.h	사용자가 액세스할 수 있는 라이브러리 함수들을 포함하며, 인터페이스 층을 구현한다. 여기서 정의된 함수들은 데이타의 버퍼링과 기록을 위해 라이브러리 내부 함수들을 호출한다.

사상(mapping)을 명세하며 문제중심의 액세스 라이브러리인 JPAL을 통하여 그들을 액세스한다. 그림 4의 추적 설명자(Trace Descriptor)는 임의의 추적양식을 EDL로 기술한 것으로 하나의 추적 설명자는 Header와 Body로 구성되며 Body는 하나 이상의 레코드 설명자(Record Descriptor)와 하나의 바인드 설명자(Bind Descriptor)로 구성된다. 추적 설명자를 포함하고 있는 파일을 EDL 소스파일이라고 한다.

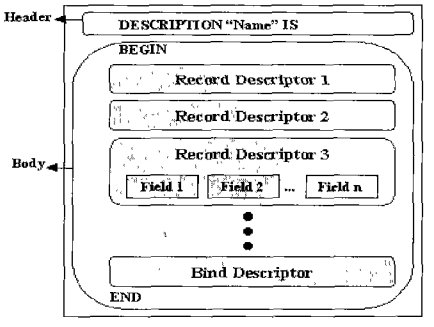

그림 4 추적설명자 구조

● 필드 설명자(Field Description) : 레코드 설명자의 각 필드의 타입과 이름을 묘사한다

```
FieldDescriptor :
    Builtin TypeSpecifications ';' ;
TypeSpecifications :
    String ArraySpecifications | String ;
ArraySpecification : { ArraySpecification ;
ArraySpecification : '[' ']' ;
Builtin :
   char | CHAR | short | SHORT | int | INT | long | LONG
   float | FLOAT | double | DOUBLE | string | STRING ;
```

그림 5 필드 설명자의 구조

● 레코드 설명자(Record Descriptor) : 논리적 레코드 타입의 이름과 구조, 그리고 표현을

나타낸다. 하나의 레코드 설명자는 하나 이상의 필드 설명자를 가지고 있다.

```
RecordDescriptor :
    RecordHeader BEGIN RecordBody END ';' ;
RecordHeader : RECORD String IS ;
RecordBody : FieldDescriptor { FieldDescriptor } ;
```

그림 6 레코드 설명자의 구조

● 바인드 설명자(Bind Descriptor) : 레코드의 구조와 표현에 관한 정보를 가지고 있는 레코드 설명자를 식별하는데 필요한 바인딩 정보를 가지고 있다. 바인딩 정보는 미리 고정된 하나 또는 두개의 키 필드(기본 키 + 보조 키) 의 값들로부터 그에 해당하는 레코드 설명자의 사상(mapping)에 의하여 표현된다.

```
BindDescriptor :
  BindHeader BindSpecifications BEGIN BindBody END ';' ;
BindHeader : BIND IS;
BindSpecifications :
  KEYFIELD INDEX IS KeySpecifications ';' :
KeySpecifications :
  '(' Number ',' Number ')' | '(' Number ',' NONE ')' |
  '(' Number ')' | Number | DEFAULT ;
BindBody : BindDescriptions ;
BindDescription : BindDescription { BindDescription ;
BindDescription : BindInformtions IS String ';' ;
BindInformations :
  '<' String ',' String '>' | '<' String '>' | String ;
```

그림 7 바인드 설명자의 구조

처리의 효율성을 위해 ASCII 형태로 기술된 추적 설명자는 EDLC(Event Description Language Compiler)라는 컴파일러를 통하여 이진(binary) 추적 설명자의 형태로 변환된다.

2.2.2 문제중심 액세스 라이브러리(JPAL)

JPAL은 추적 파일의 데이타에 대하여 문제중심의 액세스 방법을 제공하는 인터페이스 함수 라이브러리이다. 이 라이브러리는 SDDF와 유사하게 사건레코드의 논리적 구조에 기초하였으며, POET과 유사하게 문제 중심의 액세스가 가능하도록 하였다.

여기서 문제중심이란 그림 8과 같이 EDL 원시파일을 정의할 때 사용한 레코드 혹은 필드를 이름을 통하여 해당 값에 액세스할 수 있는 방법을 말한다. 그런데 JPAL을 통하여 추적 파일의 데이타를 문제중심으로 액세스하기 위해서는 앞에서 설명한 추적 설명자가 필요하다.

```
... 대신에 ...
getValue("Parent", parent);
getValue("Timestamp", timestamp);
getValue("Function", function);
... 을 사용한다 ...
```

그림 8 문제중심 인터페이스 방법

추적 설명자는 효율적인 처리를 위하여 이진 축약 형태로 존재한다. EDL 원시 파일로부터 생성된 이진 축약 형태의 추적 설명자는 EDLC 컴파일러를 통하여 생성되는데, 파일의 형태로 저장되었다가 JPAL에 의해 읽혀질 수도 있고, 파일에 저장되는 과정을 거치지 않고 EDLC를 통하여 직접 JPAL에 의해 읽혀질 수도 있다 그러나, 일반적으로 추적 양식은 안정화되어 있으므로, 처리 속도를 고려해 볼 때, 파일의 형태로 추적 양식 설명자를 저장하여 두는 것이 바람직하다. EDL/JPAL을 통하여 PICL, Gthread, XAB 등의 물리적 추적 양식으로 생성된 추적 파일을 EDL 양식에 의하여 처리하고자 할 때 그림 3과 같은 절차를 따른다 [9,10].

가시화 프레임워크는 JPAL을 통하여 추적 파일을 액세스하는데, JPAL은 이진 추적 파일로부터 읽어들인 추적 설명자를 참조하여 추적 파일을 처리한다. 이진 추적 설명자는 가시화 프레임워크의 실행에 앞서 EDL 원시 파일로부터 EDLC를 통하여 생성되어 파일의 형태로 저장되거나, 가시화 도구의 실행 시마다 파일 저장 과정을 거치지 않고 EDL 원시 파일로부터 EDLC를 통하여 번환 과정을 거친 후에 메모리로 직접 load 될 수 있다. 다음 표 2는 JPAL을 구성하고 있는 Class를 보여준다.

표 2 EDL/JPAL 인터페이스 구성 CLASS와 기능

클래스 이름	기 능
Dictionary	추적양식을 기술하는 레코드 설명자, 필드 설명자, 바인드 설명자에 대한 객체를 관리
RecordDescriptor	레코드 설명자 객체를 생성하여 생성되는 필드 설명자의 객체를 목록으로 관리
RecordDIterator	레코드 설명자를 액세스할 수 있는 방법을 제공
FieldDescriptor	필드 설명자 객체를 생성하여 정보를 관리 유지
FieldDIterator	필드 설명자를 액세스할 수 있는 방법을 제공
BindInfo	바인드 설명자 객체를 생성하여 정보를 관리하고 제공한다.
Reader	Dictionary를 사용하여 추적 파일에 기록된 사건 레코드를 반환한다.
Record	EDL Data에 대한 액세스 방법을 제공

2.3 성능 가시화 층

인스트루멘테이션 층에서 수집된 각 프로세스별 사건 추적 파일은 시간별로 정렬(sort)되어 하나의 추적파일을 이룬다. 이러한 추적파일은 가시화 층을 통하여 그림의 형태로 사용자에게 제공된다.

기존의 성능 가시화 프레임워크는 많은 문제점을 내포하고 있다. 문제점으로는 범용성의 결여 및 특정 플랫폼, 특정 도메인 또는 특정 추적양식과의 밀 결합, 물리적 추적 양식의 융통성 결여 그리고 가시화 모듈과 인스트루멘테이션 모듈간의 밀 결합이 있었다. 또한 성능 뷰의 추가 및 수정의 어려움, 성능 뷰의 규모 확장성 결여, 성능 뷰와 프로그램 동작간에 사상 메커니즘 결여, 다른 지원 도구들과의 연동 메커니즘의 부재가 있었다. 이러한 문제를 해결하기 위해 다양한 디자인 패턴을 설계에 도입했으며 성능 뷰 클래스 라이브러리를 구현하였다 [11].

2.3.1 성능 가시화 프레임워크의 설계

그림 9는 본 논문에서 제시하는 성능가시화 프레임워크의 Main Class Diagram을 나타낸다. 최고 상위레벨의 클래스로 이루어져 있으며 인스트루멘테이션 층과의 독립성을 유지하며, 임의의 응용 도메인을 위한 뷰 및 뷰 그룹의 동적 설정이 가능하고 뷰의 추가 및 수정이 용이하도록 Software-IC로 설계하였다. 이때 각 구성요소의 기능은 다음과 같다.

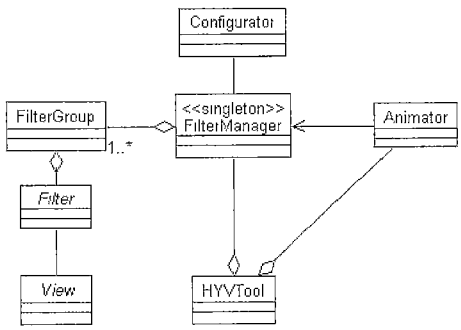

그림 9 성능가시화 프레임워크 Main Class Diagram

- 애니메이터(Animator) : 사용자가 선택한 모든 제어동작(stop, play, pause, replay, ·)을 GUI(Graphic User Interface)로부터 얻어와 Filter Manager에게 넘겨줌으로써 FilterManager는 각 제어동작에 맞게 필터를 관리한다.
- HYVTool : FilterManager와 함께 Singleton

pattern으로 설계되어 프로그램이 호출될 때 단 한번 호출되며 FilterManager, Animator 그리고 Configurator의 인스턴스를 가져온다.

- 컨피그레이터(Configurator) : FilterManager에 의하여 다루어질 FilterGroup, Filter 그리고 뷰(view)에 대하여 동적으로 구성, 관리한다.
- 필터 관리자(Filter Manager) : 성능 뷰를 구동시킬 Filter를 관리하며 사건 데이타를 전달한다. 이때 Filter를 직접 접근하는 것이 아니라 각 FilterGroup을 통하여 Filter를 관리한다.
- 뷰(View) : 뷰의 추가 및 수정이 용이하도록 Composite pattern으로 설계되었으며 프로세스의 성능 및 동작에 관한 정보를 시각화하여 보여준다.
- 필터(Filter) : 사건(event) 데이타를 분석하여 해당되는 성능 뷰를 구동시킨다.

그림 10은 성능 가시화기의 Sub-Class Diagram을 나타낸다. 그림 9의 세부 Diagram으로 FilterGroup 과 FilterManager에 적용된 패턴 및 이들간의 관계(relation)를 나타낸다.

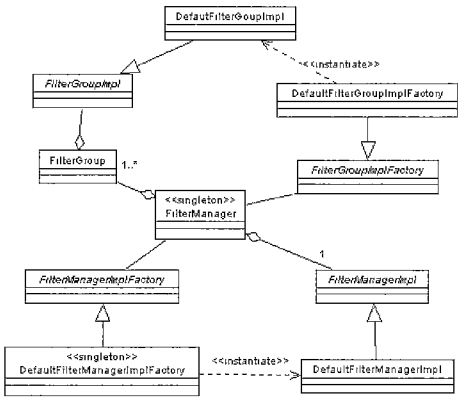

그림 10 성능 가시화 프레임워크 Sub-Class Diagram

2.3.2 패턴의 적용

- Singleton Pattern의 적용

Singleton pattern은 FilterManager와 HYVTool (HanYang Visualization Tool)에 적용된 pattern으로 singleton pattern의 적용은 한 클래스에 대하여 오직 하나의 인스턴스를 가지며 액세스할 수 있는 global point를 오직 하나만 둘 때 사용하는 패턴이다. 그림 9의 본 프레임워크에서는 프로그램이 호출될 때 인스턴스를 하나 얻어온다. 그리고 이를 통해 FilterManager와 Configurator 그리고 Animator의 인스턴스를 얻어온다. 이때 각각 하나의 인스턴스만이 필요하기 때문에 Singleton pattern을 적용하였다.

- Factory Method Pattern의 적용

Factory Method pattern은 FilterGroup과 FilterManager의 인스턴스를 생성하기 위해 적용한 패턴이다. factory method의 사용 분야는 객체 생성에 대한 인스턴스를 정의하고 그 서브클래스에서 인스턴스시킬 클래스를 결정하고자 할 때 사용하는 패턴이다. 이렇게 factory method pattern을 사용함으로써 사용자에게는 인스턴스만을 제공하고 클래스를 조작할 수 있는 방법을 원천적으로 막을 뿐 아니라 프레임워크의 확장을 원할 경우 서브 클래스만을 변경하여 사용할 수 있도록 한다.

2.3.3 추상 뷰 클래스 라이브러리의 구현

성능 뷰의 확장 및 재사용을 용이하게 하기 위해 기존 가시화 도구들이 제공하는 뷰들을 분석한 후 공통적인 특성만을 이용해 설계하였다. 이때 메시지 송수신 관계 등의 특정 뷰를 생성하려면 그림 11에서와 같이 애니메이션을 위한 추상적 뷰로부터 메시지 송수신 관계 등의 특정한 뷰를 쉽게 생성할 수 있다. 또한 모든 성능 뷰는 Software IC로 설계되었기 때문에 성능 뷰의 추가와 삭제는 다른 컴포넌트에 영향을 주지 않는다.

이러한 성능 뷰는 새로운 성능 뷰 그룹을 구성하는데 사용될 수 있다. 새로운 성능 뷰 그룹은 기존의 성능 뷰 라이브러리를 사용하여 쉽게 구성할 수 있으며 이러한

```
class Animview extends View{
        ...
        int numNodes;
        int nodeState[numNode];
    public Animview(){
        void setNumNode(int n);
        void setState(int state, String label);
        ...
    }
}
```

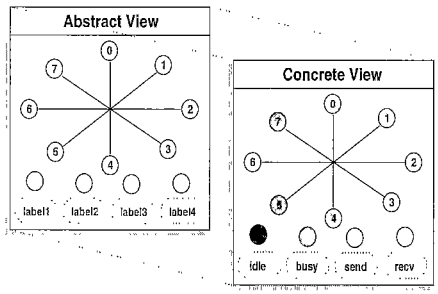

그림 11 추상 뷰 클래스의 실체화

성능 뷰의 사용은 성능 뷰 라이브러리에 필요한 데이타를 필터링을 통하여 뷰 라이브러리에 제공함으로써 사용 가능하다.

2.4 성능 가시화 프레임워크의 구현결과

본 논문에서 제시한 프레임워크는 언어와 플랫폼에 독립적인 객체지향 기법을 적용하여 설계 및 구현하였기 때문에 재 사용성을 지닌다. 또한 뷰의 추가 및 삭제가 용이하기 때문에 확장성 및 유용성을 갖는다. 그리고 객체지향 기법을 이용함으로써 기존의 성능가시화기와 차별화를 둘 수 있다.

2.4.1 성능가시화 프레임워크 구현환경

- Operating System
  Solaris 2.5.1, Solaris 2.6, Window95
- Hardware
  W/S: HYUNDAI Axil, Ultra Spack 1k
- Software
  - JDK 1.2
  - JLex 1.0.2 (Tokenizer Generator)
  - JavaCup 2.0 (Lexical analyzer Generator)

2.4.2 성능 가시화 프레임워크의 가시화 환경

확장성있는 성능 가시화 프레임워크에서 제시하는 성능 가시화 환경은 그림 12에서와 같다. 다수의 프로세스 또는 쓰레드들로 이루어진 병렬 프로그램이 수행되는 동안 인스트루멘테이션에 의하여 수집된 데이타를 사건 추적 파일(event trace file)에 추적 레코드(trace record)로 저장한다. 이렇게 저장된 정보는 가시화 층에서 원하는 사건만을 추출해내는 단계인 필터링을 통하여 유용한 정보로 변환된다. 최종적으로 가시화 단계에서 사용자에게 동적 또는 정적인 정보로 제공되어 병렬 프로그램의 동작 특성에 대한 폭넓은 통찰력을 제공한다.

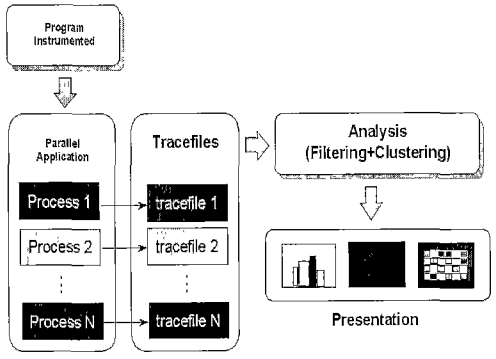

그림 12 성능 가시화 프레임워크 수행환경

2.4.3 임의의 응용 도메인을 위한 뷰 그룹 과 뷰의 동적 설정

기존의 성능가시화기는 뷰와 뷰 그룹이 경직된 구조를 가지고 있어 변경 및 설정에 있어 상당한 어려움이 있다. 따라서 새로운 뷰의 추가 및 뷰 그룹의 설정에 있어서 코드의 전체 구조를 바꾸어야 하는 어려움이 있다. 하지만 본 논문에서 제시하는 성능가시화 프레임워크는 이러한 문제점을 해결하고 이식성, 확장성, 효율성을 제공하기 위하여 계층구조를 갖도록 설계 및 구현하였다. 그림 13은 새로운 뷰의 추가 및 뷰 그룹의 변경을 동적으로 설정할 수 있도록 구성된 config 파일을 나타낸다. 새로운 뷰 그룹 및 뷰의 추가를 동적으로 설정하기를 원할 경우 config 파일에 그룹명과 그에 속한 뷰를 그림과 같이 써주기만 하면 된다. config 파일은 맨 처음에 view_group name : abstract_view name : concrete _view name 순으로 나열되며 세미콜론(:)으로 구분한다. 그림 13은 Utilization 그룹: Utilper_ViewLeaf 뷰, Util_ViewLeaf, Communication 그룹: Comm_View Leaf 뷰,Ani_ViewLeaf 뷰, Performance 그룹: Perform _ViewLeaf 뷰, Aperform_ViewLeaf, ...을 나타낸다. 또한 그림 16은 그림 13으로 세팅된 뷰 그룹 및 뷰를 타낸다.

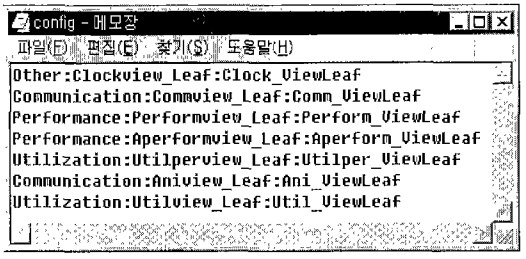

그림 13 성능 뷰 및 성능 그룹의 동적 설정

2.4.4 성능 가시화 프레임워크의 사용자 인터페이스와 성능 뷰

본 논문에서는 가시화 프레임워크의 제어 및 동작 수행을 위해 HYVTool이란 GUI(Graphic User Interface)를 제공한다. 이는 사용자의 요구에 따라 그 기능 및 형태를 얼마든지 달리할 수가 있다. 그림 14는 HYVTool의 초기화면을 나타낸 것이다. 사용의 편리성을 고려하여 동작 명령을 아이콘화 하여 구현하였다. 그림 15는 가시화가 가능한 모든 추적 형태의 파일을 나타내며 그에 해당하는 추적파일을 선택하는 부분이다. 만약 새로운 형태의 추적파일을 가시화하고 싶다면 추

적파일을 해석할 수 있는 EDL만 추가시킨 뒤 컴파일 후 쉽게 사용할 수 있다. 그림 16은 뷰 그룹 및 그에 해당하는 뷰를 선택하는 GUI를 나타낸다. 이러한 뷰의 선택은 여러 그룹의 뷰를 동시에 선택할 수 있다. 그림 17은 성능 뷰 구동시 선택할 수 있는 option을 나타낸다. 그림 18은 실제 구동되고 있는 성능 뷰를 나타낸다. PICL 형태의 추적파일에 대하여 3개의 뷰에서 동시에 수행하고 있는 부분이다.

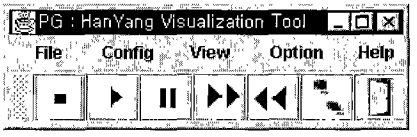

그림 14 성능 가시화 프레임워크 초기 GUI

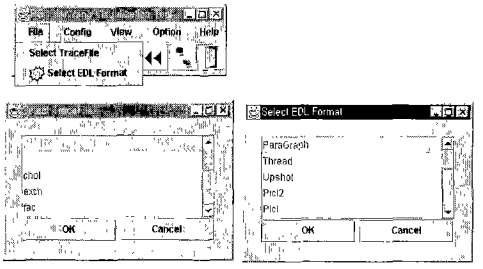

그림 15 EDL 선택과 Tracefile 선택 GUI

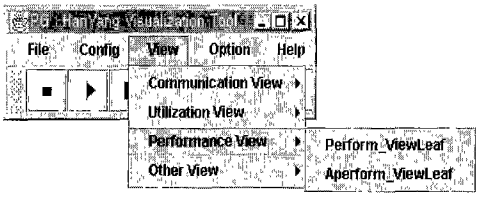

그림 16 뷰 그룹 및 뷰 선택 GUI

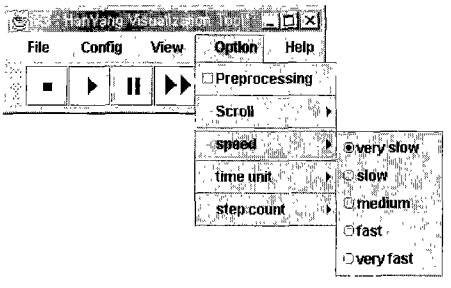

그림 17 Option 선택 GUI

이를 통해 사용자는 여러 뷰의 성능을 동시에 서로 비교, 분석함으로써 병렬 프로그램의 성능 디버깅을 쉽게 할 수 있다. 이렇게 개발된 성능 가시화 프레임워크는 인간의 이미지 처리능력을 이용함으로써 고성능 병렬 소프트웨어 개발의 기반 도구로서 뿐만 아니라 통합 병렬 프로그래밍 환경(프로그래밍, 디버깅, 성능분석) 구축의 핵심도구로 활용할 수 있다.

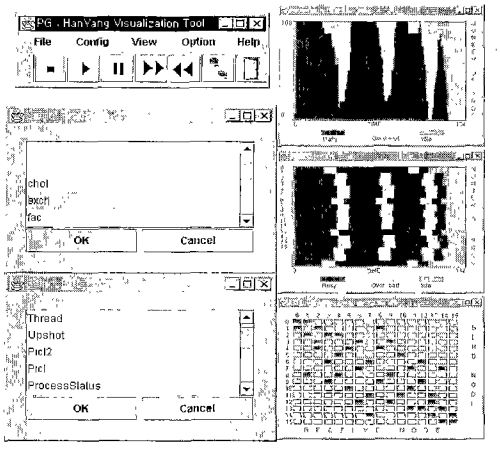

그림 18 성능가시화 프레임워크 실행모습

그림 18의 실행모습을 보면 다음과 같다. 우선 성능 측정을 위해 원하는 Tracefile(성능측정을 위한 Data 포함)을 선택하고 각각의 Tracefile에 맞게 data를 읽어들이기 위한 EDL format을 선택한다. 그런후에 그림 16으로부터 가시화 하고자 하는 형태의 뷰를 선택하고, 너무 늦게 그려질 경우 성능 측정의 어려움이 있을 수 있음으로 그려질 속도, 시간간격, 화면스크롤 단위 등 다양한 정보를 세팅한다. 이렇게 모든 준비가 완료되면 play button(▶)을 클릭하여 뷰를 동작시킨다. 첫 번째 뷰는 accumulate utilization view를 나타낸다. 이는 시간이 경과함에 따라 사용된 모든 프로세스의 상태(busy, idle, overhead)를 한데 모아 나타내며 이를 통해 전체 프로세스의 활용도를 알수 있다. 두 번째 뷰는 utilization view로 시간별로 각 프로세스의 상태(busy, idle, overhead)를 나타낸다. 세 번째 뷰는 communication view로 시간별로 각 프로세스간 data send 와 data receive 상태를 나타낸다. 붉게 칠해진 부분을 보면 보내는 프로세스와 받는 상태의 프로세스 number를 알 수 있으며 이를 통해 여러 프로세스간 통신 상태를 측정할 수 있다.

그림 19는 clock 뷰와 animation 뷰가 추가되어 동작하는 전체 실행환경을 보여준다. 하나의 tracefile을 통하여 여러 프로세스를 이용하는 parallel program이 성능 및 설계면에서 어떻게 동작하는지 visual 하게 관찰할 수 있는 실행모습이다.

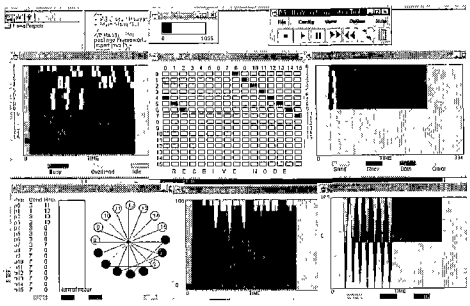

그림 19 Parallel Program의 성능측정을 위한 성능가시화 프레임워크 실행모습

3. 성능가시화 프레임워크의 비교

기존의 성능가시화 프레임워크의 종류와 그 특징을 기술한다.

3.1 PICL/ParaGraph

PICL(Portable Instrumented Communication Library)은 광범위한 메시지 전달 방식의 컴퓨터 시스템에서 효율적이고 일관된 인터페이스를 제공함으로써 이식성 높은 병렬 프로그램의 개발을 용이하게 하기 위하여 개발되었다. 이 라이브러리는 성능 디버깅을 위한 실행 추적 기능도 제공한다.

PICL에 의하여 생성된 사건 추적은 ParaGraph에 의하여 분석된다. ParaGraph는 메시지 전달 방식의 병렬 프로그램의 동작에 대한 상세한 동적 애니메이션과 성능 측정치에 대한 통계를 가시화하여 제공하는 도구이다. PICL은 소프트웨어 성능 감시 기법을 이용하는데, 메시지 전송 라이브러리를 instrument 함으로써 원래 사용자 코드를 수정할 필요가 없도록 하였다. PICL에서 사건 추적은 사건 레코드들 (메시지 송신 또는 메시지 수신등)로 구성된다. 하나의 사건 레코드는 사건의 타입, timestamp, 처리기 번호, 메시지 크기 그리고 다른 유용한 정보들을 명시하는 정수(integer)로된 필드의 집합이다. 하나의 사건 추적화일은 하나의 사건 분절만을 포함하며, 따라서 분절 헤더가 존재하지 않는다. 사건 레코드는 그 타입에 따라 고정된 수의 레코드 필드를 가지며 각 필드는 정의된 위치에 따라 고정된 의미(semantics)를 갖고 있다. 즉, PICL의 사건 추적 양식은 사건 레코드의 구조 (structure) 와 레코드 필드간의 의미 (semantics)가 물리적으로 미리 정의되어 있다.

3.2 TAU

TAU(Tuning and Analysis Utilities)는 병렬 객체지향 시스템인 pC++를 위하여 설계되고 개발된 병렬프로그램 분석 환경이다. TAU의 목적은 프로그램 분석 도구들을 위한 새로운 개발 전략의 잠재적 이익을 보여주기 위한 것이다. 새로운 개발 전략이란 범용 기능을 제공하는 전략보다는 특정한 언어의 분석 요구 조건을 만족시키도록 하는 것이다. TAU는 구현을 위해 컴파일러와 변형 도구들 (특히, Sage++ toolkit)에 의존도가 상당히 높다는 점에서 pC++ 시스템과 통합되어 있다고 할 수 있다. TAU는 다음과 같은 조건을 만족시키는 것을 목적으로 한다:

- ▶ 사용자 수준 (문제 수준)의 뷰를 제공
- ▶ 고수준의 병렬 프로그래밍 언어를 지원
- ▶ 컴파일러와 런타임 시스템을 통합
- ▶ 이식성, 확장성, 및 retargetability
- ▶ 유용성

인스트루멘테이션은 Sage++ 라이브러리의 도움을 받아 추상적 구문 트리 (Abstract Syntax Tree (AST))를 수정하는 인스트루멘터에 의해 수행된다. TAU 성능 평가 도구는 TAU 정적 도구, TAU 동적 도구 및 Pablo, SIMPLE, Upshot, POPEYE 등과 같은 외부 성능 분석 도구들로 구성된다.

TAU 정적 분석 도구들은 사용자들로 하여금 빨리 대규모 pC++ 프로그램의 개요를 얻고 그곳을 살펴볼 수 있도록 하여주는 세 가지 도구들로 구성된다: 전역 함수 및 method 브라우저(fancy (File ANd Class displaY)), 정적 callgraph 디스플레이 (cagey (CAll Graph Extended displaY)), 클래스 계층구조 디스플레이 (classy (CLASS hierarchY browser)). 동적 분석 도구들은 사용자들로 하여금 프로그램의 실행 동작을 탐험하고 분석할 수 있도록 해준다. 프로화일링 (Profiling)은 프로그램의 동작을 요약하는 통계적 정보를 계산해 줌으로써, 사용자들이 병렬프로그램의 병목 부분을 빠르게 찾아내고 주의를 집중할 수 있도록 해준다.

3.3 Voyeur

Voyeur는 병렬프로그램의 프로그램에 따른 가시적 뷰의 생성을 용이하게 해주는 프로토타입 시스템이다. Voyeur는 프로그래머가 머리속에 그리고 있는 모델과 밀접하게 연관되는 뷰를 생성하는데 촛점을 맞춤으로써, 프로그램에 대해 프로그래머가 가지고 있는 개념과 문

제 해결을 위해 고안된 실제 병렬프로그램과의 간격을 메우려 한다. Voyer는 병렬프로그램의 뷰를 생성하기 위한 언어 독립적이고 시스템 독립적인 메카니즘을 제공한다. Voyer는 계산 도메인의 이미지를 보여주는 상당히 추상적인 뷰에서, 단지 프로그램 변수들을 보여주는 언어 고유의 혹은 아키텍춰 고유의 뷰에 이르기까지 계층적인 뷰를 지원하고 있다.

3.4 Pablo

Pablo는 규모확장성이 있는 다양한 병렬시스템에서의 성능 데이타 포획, 분석 및 가시화를 제공하기 위해 설계된 성능 분석 환경이다. Pablo의 주요 설계 목적은 이식성, 규모 확장성 (scalability) 및 확장성 (extensibility)에 있다. Pablo는 성능 분석 환경의 구축을 위한 도구로 보는 것이 합당하다. 그것은 (1)이식성 있는 소프트웨어 인스트루멘테이션과 (2)인스트루멘테이션을 데이타 분석과 연결시키는 추적 데이타 메타-양식을 가지고 있는, 이식성 있는 성능 데이타 분석이라는 두개의 주요 요소들로 구성된다. Pablo의 이식성 있는 소프트웨어 인스트루멘테이션은 소스 코드 인스트루멘테이션 지점에 대한 대화식 명세를 지원하도록 설계되었다. 인스트루멘테이션 명세를 위한 그래픽 인터페이스, 인스트루멘트된 코드를 생성하는 변형된 C 및 Fortran parser, 그리고 추적 포획 라이브러리들은 Pablo 소프트웨어 인스트루멘테이션의 세 가지 중요한 요소이다. Pablo의 성능 분석 요소는 그래픽의 형태로 상호 연결되어 비순환적 데이타 분석 그래프를 형성하는 데이타 변환 모듈의 집합으로 구성된다. 성능 데이타는 그래프 노드들을 통해 흘러가며, 원하는 성능 metrics를 생산하도록 변형된다. 각 성능 변형 모듈은 데이타 변형 코아 (core) (즉, 실제 데이타 변형)와 시스템이 제공하는 데이타 엑세스 wrapper로 구성된다. 데이타 엑세스 wrapper는 데이타 타입, 크기, 이름 등의 내부 정의들 포함하지만, 미리 정의된 의미 (semantics)를 포함하지 않는 self-documenting 데이타 스트림 메타-양식을 읽고 쓰도록 해준다. Wrapper의 데이타 엑세스 방법은 데이타 스트림의 구조를 미리 알 필요 없이, 데이타 필드의 정보를 추출하거나 생성할 수 있도록 하고, 그와 관련된 소프트웨어는 데이타 복제와 다른 노드로의 배분을 지원한다.

3.5 XAB

XAB(X-window Analysis and deBugging)는 PVM (Parallel Virtual Machine) 프로그램의 성능감시를 위한 도구이다. XAB를 사용함으로써 PVM 프로그램을 쉽게 감시할 수 있다. XAB는 PVM 프로그램을 감시할 목적으로 PVM을 사용한다. XAB는 세 가지 기본 요소들로 구성된다: 사용자 라이브러리, 감시 프로그램, 그리고 X 윈도우 frontend. 사용자 라이브러리는 인스트루멘트된 PVM 호출을 제공한다. 감시 프로그램은 PVM 프로세스로서 수행되면서 PVM 메시지 형태의 사건들을 감시하고 수집한다. XAB frontend는 PVM 프로세스들과 메시지에 대한 정보를 그림으로 표시해 준다. XAB는 PVM 라이브러리에 대한 호출을 인스트루멘트함으로써 PVM 프로그램을 감시한다. 인스트루멘트된 호출은 프로그램이 수행되는 동안 디스플레이될 수 있는 사건들을 발생한다. PVM의 호출은 등가의 XAB 라이브러리 루틴들에 대한 호출로 대치됨으로써 인스트루멘트된다. 프로그래머는 PVM을 호출하는 소스 화일에 xab.h 헤더 화일을 내포해야 한다. 이 헤더 화일에는 원래의 PVM 호출을 XAB 라이브러리에 대한 호출로 대치하도록 하는 매크로들이 포함되어 있다. XAB 루틴들은 사용자를 위해 정상적인 PVM 기능을 수행하면서, 특별한 감시 프로세스 (abmon)에 PVM 메시지를 전송한다. abmon 프로세스는 인스트루멘트된 PVM 호출로부터 메시지를 전달받으며, 그것을 사람이 읽을 수 있는 형태로 양식화한다. 디스플레이 프로세스는 양식화된 사건을 취하여 실시간에 그림으로 디스플레이 한다. XAB의 실시간 디스플레이 기능의 장점 중에 하나는 사건이 발생하는 즉시 그것을 보여줄 수 있다는 것이다.

3.6 VISTOP

가시화 도구인 VISTOP (VISualization TOol for Parallel systems)은 MMK 병렬프로그래밍 라이브러리로 개발된 메시지 전달 프로그램을 에니메이션 한다. VISTOP은 기본적인 MMK 객체 타입인 태스크, 메일박스, 세마포를 통해 응용프로그램의 구조를 가시화한다. 따라서, VISTOP은 MMK 프로그래밍 모델을 사용하는 프로그램의 추상화된 수준의 에니메이션을 보여준다.

3.7 IPS-2

IPS-2는 병렬프로그램 성능 분석 도구로서, 공유 메모리 환경이나 메시지 전달 환경에서 수행되는 응용프로그램의 성능을 감시하고 분석하는데 사용된다. 응용프로그램의 성능은 프로그램 전체로부터 각각의 프로시쥬어 혹은 동기화 변수에 이르기까지 다양한 수준에 걸쳐 상세하게 연구될 수 있다. 추적된 사건들은 프로시쥬어 entry/exit, 동기화 연산, 프로세스 연산, 그리고 I/O 연산 등이다. IPS-2는 사용자에게 프로그램에 관한 정보를 계층적 형태로 제시한다. 이때 지원되는 계층은 네 단계로 구성된다: 프로그램 수준, 기계 수준, 프로세스 수준, 그리고 프로시쥬어 수준. 새로운 버젼의 IPS-2 병

렬프로그램 측정 도구는 응용프로그램, 운영체제, 하드웨어, 네트워크 등으로부터의 성능 데이타를 제공한다. 응용프로그램과 시스템을 위한 측정 기준을 통합하는 것은 "외부 시간 히스토그램 (external time histogram)" 이라는 개방형 인터페이스를 개발함으로써 가능하게 되었다. 외부 시간 히스토그램은 다양한 공급원으로부터의 외부 데이타들을 매끄럽게 포함할 수 있는 방법을 제공한다. 각각의 시간 히스토그램은 시간에 대한 하나의 성능 측정 기준의 값을 묘사하는 자료구조이다. 외부 metrics를 포함함으로써, 프로그램의 성능 분석에 사용될 수 있는 정보의 유형에 새로운 차원이 첨가될 수 있다. IPS-2는 자동적으로 프로그래머를 프로그램의 병목위치로 안내하도록 도와 주며 규칙적이고, 계층적으로 분할 수 있는 대부분의 시스템에 적용할 수 있다. 프로그램의 계층구조는 분산 프로그램의 다양한 수준의 추상화를 보여주기 위한 일관된 framework를 제공한다. 병렬프로그램의 성능을 이해하고 싶다면, 우리는 그 동작을 각기 다른 수준에서 상세하기 관찰할 수 있다. IPS-2에서는 병렬프로그램의 계층구조에 따른 계층적 측정 구조를 선택하였다. 계층구조의 각 레벨에서 프로그램의 수행을 묘사할 수 있는 성능 metrics를 정의한다.

3.8 병렬프로그램의 개발을 지원하기 위한 가시화 도구의 특성

- ▶ 디버깅을 가시화하기 위한 도구는 응용프로그램에 재빨리 적응
- ▶ 소스코드보다는 목적코드의 인스트루멘테이션이 바람직(에러의 확률 감소).
- ▶ 가시화 도구는 on-line 대화식으로 사용
- ▶ 가시화 도구는 병렬프로그래밍 환경의 다른 도구들과 통합: 예, 소스 수준의 디버거
- ▶ 가시화 도구는 저변의 프로그래밍 모델의 추상화를 포함하여 다른 수준의 추상화를 지원
- ▶ 가시화 도구는 MPP 시스템의 응용프로그램에도 적용가능 해야함.
- ▶ 가시화 도구는 응용프로그램을 위한 복수의 뷰와 추상화를 제공해야함, 이는 사용자로 하여금 응용프로그램을 각기 다른 측면에서 분석할 수 있도록 해줌
- ▶ 가시화 도구는 동적 프로세스 모델을 지원해야 하며 시스템에 통합된 자동적 부하 균형의 동작을 보여줄 수 있어야 함.
- ▶ 가시화 도구는 확장성이 있어야 하며 만일 사용자가 프로그램에 관한 추가적인 정보를 얻기를 원한다면, 그것은 가시화 도구의 최소한의 수정을 통해 가능하도록 함.

다음 표 3은 기존의 성능 가시화기와 본 논문에서 제시한 HYVT의 가시화 측면에서 비교한 결과를 보여준다. 비교항목은 static, animation, summary, profile, layered architecture 의 5가지 항목에 대하여 비교하였다. static 항목은 특정언어만의 분석요구를 만족시키는지를 나타낸다. 즉, Static한 성격을 가진 성능가시화기는 특수한 언어에만 적용가능하며 다른 언어에는 사용할 수 없는 특성을 지닌다. TAU의 경우 병렬 객체지향 시스템인 pC++을 위하여 설계되었다. Animation은 병렬프로그램의 성능분석을 Graphic하게 측정할 수 있는가를 나타낸다. Summary는 최종 수행결과에 대한 성능을 총괄적으로 표현할 수 있는가를 나타낸다. Summary를 할 수 없는 가시화기는 병렬프로그램에 대한 최종 성능을 통계적으로 나타낼 수 없어 그 성능측정에 어려움을 갖는다. Profile은 수행할 뷰에 대하여 추상(Abstract) 뷰를 제공하는지를 나타낸다. 이는 사용자 수준(문제수준)의 다양한 뷰를 제공할 수 있는지에 대한 기능으로 뷰의 확장성과 재사용성이 용이한가를 나타낸다. Profile이 가능한 가시화기는 특정한 뷰를 생성하고자 할 때 추상 뷰로부터 구체적(Concrete)뷰를 쉽게 생성할 수 있다. 마지막으로 Layered architecture 항목을 설정하였다. 이는 프레임워크를 각각 독립적으로 개발가능한 계층구조를 둠으로써 범용성, 확장성, 이식성을 제공할 수 있는가를 나타낸다. 이는 만일 사용자가 성능측정을 위한 시스템이나 언어가 바뀌든지 성능측정에 대한 추가적인 기능을 삽입하기 원할 때 가시화 도구의 최소한의 수정을 통해 적용가능한지를 나타낸다. 이를 통해 응용프로그램에 빠른 적용이 가능하다.

표 3 성능가시화 프레임워크의 비교(가시화)

	Visualization				
Name	Static	Animation	Summary	Profile	Layered Architecture
ParaGraph		X	X	X	
TAU	X	X	X	X	
Voyeur		X			
Pablo		X			
XAB		X			
VISTOP		X			
ISP-2		X	X	X	
HYVT		X	X	X	X

4. 결론 및 향후 과제

본 논문에서 제시하는 성능 가시화 프레임워크는 디자인 패턴의 적용과 객체지향 설계 및 프로그래밍 기법을 이용해 각각 독립적으로 개발 가능한 계층구조로 구현함으로써 기존의 가시화기에서 가지고 있던 문제점인 범용성, 확장성 그리고 이식성의 결여를 해결할 수 있었다. 또한 뷰와 가시화 프레임워크간의 밀 결합으로 인해 발생한 수정 및 변형의 어려움을 해결하기 위하여 추상 뷰(abstract view) 클래스 라이브러리를 구현하였다. 최종적으로 성능 가시화 층은 애니메이터, HYVTool, 컨피그레이터, 필터관리자 등으로 구현하였으며 가시화에 필요한 backbone 아키텍춰를 제공함으로써 plug-and-play 방식으로 동작하도록 하였다. 처음 사용하는 사용자를 위해 MPI 프로그램의 동적 동작 특성과 성능에 관한 다양한 정보를 제공하기 위하여 다양한 뷰를 디폴트로 제공 할 수 있었다. 향휴 연구 과제로는 본 논문에서 제시한 프레임워크에 back-tracking 기능과 텍스트-그래픽 사상(mapping) 메카니즘 같은 다양한 기능을 포함시킬 계획이다.

참 고 문 헌

[1] John Stasko, John Domingue, Marc H. Brown and Blaine A. Price, "Software Visualization," MIT Press, 1998.

[2] M. T. Heath and J. A. Etheridge, "Visualizing performance of parallel programs," Technical Report ORNL/TM-11813, Oak Ridge National Laboratory, Oak Ridge, TN, May 1991.

[3] Michael T. Heath, Jennifer A. Etherridge, "Visualizing he performance of parallel programs," IEEE Software, ep. pp.29-39, 1990.

[4] J. Kim and C. T. Wright, "An object-oriented approach towards a general purpose performance visualization for parallel programs," second annual Midwest ElectroTechnology Conference, pp. 6-9, April, 1993.

[5] M. Simmons and R. Kokela, editors, "Performance instrumentation and visualization," New York: ACM & Addison-Wesley, 1990

[6] Message Passing Interface Forum, "The MPI message passing interface standard," Tech. Report, University of Tennessee Kenoxville, April 1994. Available form http://www.msc.anl.gov/mpi/mpi-report.ps

[7] 진휴정, 김종렬, 김정선, 문영식, "MPI 프로그램을 위한 성능 분석 시스템," 한국정보과학회 추계학술대회 발표논문집 제24권 3호, pp 467-470, 1997.

[8] R. A. Aydt, "The pablo self-defining data format," University of Illinois Board of Trustees, March 1992.

[9] A. Beguelin, "Xab: a tool for monitoring PVM programs," Workshop on Heterogeneous Processing, Los Alamitos, Ca., April, 1993.

[10] G. A. Geist, M. T. Heath, B. W. Peyton and P. H. Worley, "Auser's guide to PICL: a protable instrumented communication library," Oak Ridge National Laboratory, Oak Ridge, TN, Technical Report ORNL/TM-11616, August, 1990.

[11] Erich Gamma, Richard Helm, Ralph Johnsom and John VLissides, "Design Patterns Elements of Reusable Object-Oriented Software," Addison Wesley, 1994.

문 상 수

1998년 서원대학교 컴퓨터 공학과 학사. 2000년 한양대학교 컴퓨터 공학과 석사. 2000년 (주)엠헨즈 기술 및 총괄이사. 2000년 (주)엠헨즈 대표이사. 관심분야는 병렬처리, 시스템 보안, 객체지향 개발 방법론, 소프트웨어 공학.

문 영 식

1980년 2월 서울대학교 공과대학 전자공학과 졸업(학사). 1982년 2월 한국과학기술원 전기 및 전자공학과 졸업(석사). 1990년 University of California at Irvine Dept. of Electrical and Computer Engr. (박사). 1982년 ~ 1985년 한국전자통신연구소 연구원. 1989년 ~ 1990 Inno Vision Medical 선임연구원. 1990년 ~ 1992년 생산기술연구원 선임연구원. 1992년 ~ 현재 한양대학교 전자계산학과 부교수. 관심분야는 영상처리, 컴퓨터 비전, 패턴인식, 병렬처리 등.

김 정 선

1986년 서울대학교 컴퓨터공학과 졸업(학사). 1988년 Iowa Statr University 전기 및 컴퓨터 공학과 졸업(공학석사). 1994년 Iowa State University 전기 및 컴퓨터 공학과 졸업(공학박사). 1994년 ~ 1996년 한국전자통신연구원(ETRI) 선임연구원. 1996년 ~ 현재 한양대학교 전자계산학과 조교수. 관심분야는 Parallel & Distributed Processing, Component Based Development, Distributed Object Computing.