최 종 연 구 보 고 서

실시간 분산시스템 소프트웨어 개발

주관연구기관 : 한국전자통신연구소

과 학 기 술 처

과 학 기 술 처 장 관 귀 하

실시간 분산시스템 소프트웨어 개발 과제의 최종보고서를 별첨과 같이 제출합니다.

1992. 07.

주관연구기관 : 한국전자통신연구소

총괄연구책임자 : 박 세 영 (인)

주관연구기관장 : 양 승 택 (인)

여 백

제 출 문

과학기술처장관 귀하

본 보고서를 "실시간 분산시스템 소프트웨어 개발" 과제의 최종보고서로 제출합니다.

1992. 07.

주관연구기관명 : 한국전자통신연구소

총괄연구책임자 : 박 세 영

선임연구원 : 임 헌 규

연구원 : 최 동 시

여 백

요약문

I. 제목

실시간 분산시스템 소프트웨어 개발에 관한 연구.

II. 연구개발의 목적 및 중요성

본 연구의 목적은 분산, 병렬시스템상에서 병렬처리 프로그램 수행시 각 단위 수행모듈들에게 프로세싱 노드를 할당할때 프로세싱 노드들 사이의 통신 Overhead를 최소화할 수 있는 프로세싱 노드의 연결관계 테이블을 자동으로 생성함으로써 프로그램의 성능을 극대화할 수 있는 실시간 분산시스템 소프트웨어를 개발하는 것이다.

이러한 시스템을 개발함으로써 멀티미디어와 같은 대용량의 데이타 처리 응용프로그램, 음성 및 패턴 인식등의 여러분야에 필요한 핵심기술을 제공할 수 있고, 차세대 컴퓨터에 대한 제반기술을 또한 제공함으로써 국내 기술 확립과 국제 경쟁력 강화에 기여할 수 있다.

III. 연구 개발의 내용 및 범위

본 연구에서는 시스템 개발의 초기 단계로서 대내외적으로 기존에 발표된 논

문이나 실제로 구현된 분산, 병렬처리 시스템등의 사례조사와 기본개념을 바탕으로 개발하고자 하는 시스템 소프트웨어의 필요성 및 타당성을 분석, 정당화 함으로써 개념모델을 정립하고 시스템 소프트웨어의 각 모듈별 설계 및 앞으로의 연구 방향 결정에 필요한 기초적인 연구를 수행하였다.

- 기본 개념

대용량의 데이타 처리 프로그램의 분산, 병렬처리를 위한 시스템 소프트웨어에 대한 연구 및 본 과제의 최종목표인 시스템 소프트웨어에 대한 필요성과 개발방법론에 대한 연구.

- 개념 모델

멀티미디어와 같은 대용량의 데이타를 처리하는 프로그램의 병렬처리를 위한 타스크 분산 맵 자동 생성 시스템 소프트웨어 개념 모델에 대한 연구.

- 시스템 스펙

개발하고자 하는 시스템 소프트웨어의 전체 분산, 병렬시스템 측면에서의 시스템 구성 요소 및 역할에 대한 연구.

- 기능 규격 설정

시스템 소프트웨어를 구성하는 각 모듈별 기능 설정에 대한 연구.

IV. 연구결과 및 활용에 대한 건의

가. 주요 연구 결과

- 분산, 병렬시스템의 사용 프로그램언어 및 운영체제에 관한 기술문서.
- 분산, 병렬시스템 소프트웨어의 기본개념 문서.
- 분산, 병렬시스템 소프트웨어의 개념 모델 문서.
- 분산, 병렬시스템 소프트웨어의 시스템 스펙 문서.

나. 활용에 대한 건의

본 과제에서 연구되고 있는 내용 및 문서등의 결과는 어떠한 특정 운영체제나 응용소프트웨어에 국한된것이 아니라 일반적으로 사용되는 하드웨어나 소프트웨어에 쉽게 구현될 수 있기 때문에 분산, 병렬 특성을 요구하는 분야 특히, 멀티미디어 데이타 처리를 위한 응용소프트웨어나 음성 및 패턴인식등의 연구에 활용될 수 있다.

V. 연구 일정

구분 / 연구내용	연구개발기간 '91 8	9	10	11	12	'92 1	2	3	4	5	6	7	진도율 (%)
1. 기본개념	◀	———	▶										100
2. 개념모델			◀	———	———	———	▷						100
3. 시스템 스펙						◁	———	———	———	▶			100
4. 기능규격									◀	———	———	▶	100

SUMMARY

Most application programs which are executed on the multiprocessor system require a lot of communication cost between processing nodes. The communication cost of a program is a very important factor for the overall program performance. When tasks of a parallel processing program are distributed and runned, there are two important issues which we must consider in order to maximize the program and system performance. One of them is how to balance the load of processing nodes in order to maximize the system performance. And the other is how to minimize the communication cost of a program.

In this work, we propose a model which can find a task distribution network to minimize the overall communication cost of an application program. The task distribution network is a physical connection of all processing nodes that are assigned to execute an application program, and the network is possible to be modified to reduce the communication cost.

Each processing node of the multiprocessor system has the limited number of links to be connected to other processing nodes, and consequently the processing nodes can not be fully connected by physical links with others. For that reason some communications between two processing nodes must be performed with a number of intermediate processing nodes, and the

communication cost over a program may be increased. Accordingly, the overall performance of a program is varied with the results of distributing tasks over processing nodes. A solution to the above variation of performance is that each program has its own task distribution network which is used at processor allocation time. But there are some problems. If the target system which is not the same hardware configuration as an application program was developed and the real world data that is handled by an application program in an actual site is not matched with the sample data by which the program was tested at developing the program, the task distribution network may not be appropriate.

In order to overcome the above problem, we propose a method that adaptively regenerates the task distribution network for the target system. When a program is runned at an actual site, we can collect the statistic data of the communication request events for the given time and regenerate the task distribution network by analyzing the collected data. Then the new network will be used at the next execution of the program. Above steps will be repeated until the gain of performance is saturated. When these steps are performed users of this program must submit themselves to the processing overhead. But this overhead will be removed after the gain of performance is saturated.

CONTENTS

Chapter 1. Introduction 1

Chapter 2. Charateristics of Multimedia Data Processing 9

Section 1. Modularization of Programs 11

Section 2. Large Volume Data Processing 12

Section 3. Synchronization 13

Section 4. Real-Time Processing 13

Section 5. Parallel Processing Features 14

Chapter 3. Multiprocessor System 17

Section 1. Basic Concept of Mutiprocessor System 20

1. Link Bandwidth 21

2. Network Configuration 21

3. Interprocessor Communication 22

4. I/O Interface 22

Section 2. Multiprocessor System Hardware 23

1. Message Passing Multicomputer 24

2. Network Topologies 25

3. Latency 25

4. Processing Unit 25

5. Communication Unit 26

6. Communication Mechanism of Transputer 27

Section 3. Distributed and Parallel Operating System Helios 28

1. Basic Concept and Internal Structure of Helios 29

2. Parallel Programming on Helios 33

3. Improvement 35

Section 4. Parallel Operating System TRACOS 36

1. TRACOS 36

2. Communication on Transputer Network 37

3. Event-Driven Monitoring 42

4. Evaluation Performance of TRACOS 46

5. Improvement 53

Chapter 4. Distributed and Parallel Language CDL 55

Section 1. CSP Model 57

Section 2. CDL 60

1. Task Force 60

2. Parallel Constructors for Task Force 61

3. Streams 64

4. Component Declaration 66

5. Replicators 67

Chapter 5. Distributed and Parallel Processing of Programs 69

Section 1. Basic Concept 71

1. Component of Parallel Program 71

2. Parallel Processing Program 72

3. Intertask Communication Network 72

4. Processing Node Allocation 73

Section 2. Intertask Communication 74

1. Delayed Data Transfer 75

2. Routing Overhead of Intermediate Nodes 76

3. Limitation of Link Bandwidth 77

Section 3. Performance Evaluation of Programs 78

1. Performance vs. Processor Allocation 78

2. Hardware Dependency 82

3. Communication Cost vs. Program Performance 82

Chapter 6. Performance Evaluation of Distributed and Parallel Search Program 85

Section 1. Test Method of Distributed and Parallel Program 88

Section 2. Distributed and Parallel Processing of Search Program 91

Section 3. Evaluation of Results 92

Chapter 7. Adaptive Task Distribution Network Reconfiguration Model 93

Section 1. Conceptual Structure 99

Section 2. Requirement of Multiprocessor System 103

Section 3. Communication Request Monitor 103

Section 4. Communication Pattern Analyzer 107

Section 5. Task Distribution Network Generator 108

1. Processing Node Allocation 111

2. Linking Two Processing Nodes 111

3. Checking Communication Link Bottleneck 113

4. Removing the Communication Bottleneck 114

5. Communication Bottlenecck Monitoring 115

Section 6. Experimental Results 116

Chapter 8. Conclusion 119

References 123

Appendix 129

목 차

제 1 장 서론 1

제 2 장 멀티미디어 데이타 처리 프로그램의 특성 9

제 1 절 프로그램의 Module화 11

제 2 절 대용량의 데이타 처리 12

제 3 절 미디어들 사이의 동기화 13

제 4 절 실시간 처리 13

제 5 절 병렬처리 특성 14

제 3 장 분산, 병렬처리 시스템 17

제 1 절 분산, 병렬처리 시스템의 기본개념 20

1. 링크 Bandwidth 21

2. 네트웍 구성 21

3. 처리기 사이의 통신 22

4. 입출력 장치 22

제 2 절 병렬처리 시스템 하드웨어 23

1. 메시지 전달 방식 다중 처리기 24

2. Topologie의 종류 25

3. 대기시간 25

4. 처리 Unit (T800) 25

5. 통신 Unit 26

6. 트랜스퓨터의 통신기능 27

제 3 절 분산, 병렬 운영체제 Helios 28

1. Helios의 기본 개념 및 내부 구조 ····· 29
2. Helios에서의 병렬처리 프로그래밍 ····· 33
3. 개선점 ····· 35
제 4 절 분산, 병렬 운영체제 TRACOS ····· 36
1. TRACOS의 필요성 ····· 36
2. *Transputer-Networks* 상의 통신체제 ····· 37
3. Event-Driven 방식의 감시 장치 ····· 42
4. TRACOS의 성능평가 ····· 46
5. 개선점 ····· 53
제 4 장 분산, 병렬언어 CDL ····· 55
제 1 절 CSP 모델 ····· 57
제 2 절 CDL 언어 ····· 60
1. Task force 실행방법 ····· 60
2. Task force 정의를 위한 parallel constructors ····· 61
3. 스트림 ····· 64
4. Component 선언문 ····· 66
5. 중복 명령어 ····· 67
제 5 장 프로그램의 분산 병렬처리 ····· 69
제 1 절 분산 병렬처리의 기본 개념 ····· 71
1. 분산 병렬처리 단위 ····· 71
2. 병렬처리 프로그램 ····· 72
3. 타스크 간의 통신망 ····· 72
4. 프로세싱 노드 할당 ····· 73

제 2 절 데이타 통신 특성 74
1. 데이타 전송 지연 시간 75
2. 중간 노드의 라우팅 Overhead 76
3. 링크 Bandwidth의 한계치 도달 77
제 3 절 프로그램의 성능분석 78
1. 프로세서 할당과 프로그램 성능 78
2. 프로그램의 하드웨어 Dependency와 문제점 82
3. 데이타 통신과 시스템 성능과의 관계 82
제 6 장 분산, 병렬 검색 프로그램의 성능 분석 85
제 1 절 분산 병렬처리 시험 방법 88
제 2 절 분산 병렬처리 방법 91
제 3 절 분산 병렬처리 결과 분석 92
제 7 장 적응 분산 맵 자동 생성 모델 97
제 1 절 시스템 구성도 99
제 2 절 병렬처리 시스템 요구사항 103
제 3 절 통신 요구 감시기 103
제 4 절 통신 패턴 분석기 107
제 5 절 타스크 분산 맵 생성 108
1. 프로세싱 노드 할당 111
2. 두개의 프로세싱 노드 연결 111
3. 통신 링크의 병목상태 추출 113
4. 통신 링크 병목 현상 제거 114
5. 통신 링크 병목 현상 감시 115

제 6 절 시험 결과 ····· 116

제 8 장 결론 ····· 119

참고문헌 ····· 123

Appendix ····· 129

제 1 장 서 론

여 백

제 1 장 서 론

산업사회로 부터 정보화사회로 변화함에 따라 사회의 생활구조가 달라지고, 사회적인 요구사항 또한 변화하게 되었다. 이러한 변화에 따라 사회생활에 필요한 정보는 날로 전문화, 다양화 그리고 복잡화 되어가고 있다. 점점 방대해지고, 복잡한 정보를 처리하기 위하여 컴퓨터의 사용은 개인용 컴퓨터의 보급확대와 더불어 대중화되고 있으며, 컴퓨터의 활용이 필수적인 분야 또한 다양해지고 있다. 따라서 이제 컴퓨터는 수치계산이나 일부 특정 응용분야 뿐만아니라 사회에서 발생되는 여러가지 문제와 요구를 처리해야 하는 과제를 갖게 되었으며, 이에 따라 컴퓨터의 기능적인 요구사항도 변화하게 되었다.

이러한 다양하면서도 복잡성을 띄는 사용자들의 요구사항을 만족 시키기 위하여 기존에 사용하던 Sequential Machine 으로는 여러가지 측면에서 많은 문제점들이 발생하였다. 최근 VLSI 기술의 발전으로 그전의 미니급 컴퓨터와 버금가는 성능을 지닌 마이크로 프로세서를 수십개 내지는 수백개를 동시에 연결시켜 시스템 전체의 성능을 향상 시켜주는 실시간 분산, 병렬처리 컴퓨터의 개발이 대내외적으로 활발하게 진행되고 있다.

그러나 초기의 분산, 병렬처리 시스템에 대한 연구는 여러개의 프로세서들을 연결하여 처리 속도를 증가시키려는 하드웨어 측면에 중점적으로 집중되었으나, 시스템 소프트웨어 측면 즉, 사용자의 응용프로그램의 성능을 증가시켜주는 방향의 연구, 다시 말해 사용자에게 시스템과 프로세서간의 연결관계, 통신상태 등을 고려하지 않고도 시

스템 자체의 성능을 극대화할 수 있는 Transparency를 제공하고, 응용프로그램의 이용도를 높이는 측면에서는 아직 미진한 상태이다.

현재 병렬처리 시스템들이 여러 분야에서 많이 사용되고 있으며, 특히 최근에 주요 연구분야 중의 하나인 멀티미디어 데이타 처리를 위한 시스템 하드웨어 및 소프트웨어에 대한 연구가 활발히 진행중이며, 기존의 시스템으로는 멀티미디어 데이타 처리를 위한 성능 및 제공기능이 빈약한 관계로 현재까지는 개인용 컴퓨터 및 워크스테이션 등에서의 멀티미디어 데이타 처리를 위한 시스템 개발에 집중되고 있다. 이러한 멀티미디어 데이타 처리를 위한 소프트웨어의 가장 큰 문제점 중의 하나는 처리 속도를 어떻게 만족시켜 주어야 하는가 이다. 이러한 처리 속도에 있어서의 문제점을 해결할 수 있는 방법중의 하나는 여러개의 프로세싱 노드로 구성되는 병렬처리 시스템의 이용이다[Patton, Lo, Gajski].

멀티미디어 데이타 처리를 위한 프로그램등과 같은 통신양이 많은 프로그램의 개발시에 최소한으로 요구되는 수행 속도 뿐만 아니라 가능한 한 빠른 시간내에 수행을 완료하도록 하기 위하여 병렬처리를 하고자 할때[Ackerman], 각 Task들 사이의 연결도(Communication Network Configuration)에 따른 최상의 Data Communication Network을 찾아내기 위한 많은 노력이 집중되고 있으며, 또한 프로그래밍 언어에 대한 연구를 수행하고 있는 엔지니어들도 병렬 프로그램 언어에 대한 연구에 집중되고 있다.

특정 응용분야에 적절한 Data Communication Network을 결정하는 것은 프로그램의 설계시에 충분히 고려되어야 하는 것으로서, 해당 프로그램에서 처리되는

데이타의 일반적인 형태와 처리 방법에 따른 것이며 또한 개념적인 통신 네트워크일 뿐 실제 각각의 병렬처리 단위 모듈들이 프로세싱 노드를 할당받아 수행될 때는 적용되지 않는다. 즉 한 응용 프로그램에서 사용된 Data Communication Network가 프로세싱 노드 할당시에 참조되지 않고 시스템 소프트웨어에서 임으로 할당되고, Network Node들 사이의 통신은 시스템 소프트웨어에서 제공하는 통신기능을 이용하게 된다.

그렇지 않고 프로그램에서 프로세싱 노드의 할당을 지정할 수도 있는데 이를 위해서는 프로그래머가 수행되는 시스템의 하드웨어 구성 및 세부 동작에 대한 지식이 있어야 한다. 또한 이러한 경우 프로그래머는 통신을 위한 기본적인 함수들을 직접 Coding해야 하며 각 프로세싱 노드에서 수행되는 Task들 사이의 통신 Path에 대한 지정이 요구된다. 이와 같은 프로그래밍의 어려움 때문에 많은 시스템에서는 개념적으로 모든 Task들 사이의 link가 존재하여 통신이 가능한 것으로 생각하고 프로그래밍할 수 있도록 시스템 소프트웨어 차원에서 통신 기능을 제공하고 있다.

병렬수행이 가능한 여러개의 Task로 구성된 프로그램은 수행시에 프로세싱 노드를 할당받게 되고 각각의 Task는 동시에 처리된다. 그러나 이러한 병렬처리에 있어서 발생되는 주요 문제점 중의 하나는 각 Task에 할당된 프로세싱 노드들은 다른 노드와의 통신을 위해 제한된 수의 link를 가지고 있는 것에서 발생된다. 통신이 필요한 각 Task들 사이를 link로 연결해보면 하나의 Data Communication Network를 구할 수 있는데 이 네트워크는 매우 복잡한 하나의 그래프로써 보통 프로세싱 노드가 갖고 있는 link의 수보다 많은 수의 link가 각 Task에 존재하게

된다. 즉 병렬처리를 위해 프로세싱 노드를 할당할 때 각 Task의 모든 link를 physical link로 연결할 수 없게 된다. 따라서 어떤 두 Task 사이의 데이타 통신을 위해서는 중간 노드를 거쳐야 하며 이러한 중간노드를 거치는 과정에서 통신 overhead는 증가되며 프로그램의 전체적인 Performance도 떨어지게 된다.

현재 대부분의 병렬처리 시스템에서는 이와 같은 통신 overhead를 최소화할 수 있도록 프로세싱 노드의 할당이 불가능 하다. 즉 이러한 통신 overhead는 응용 프로그램에 따라 많은 차이가 있으므로 시스템 소프트웨어에서는 일반적인 통신 overhead를 예측할 수 없다. 따라서 통신 overhead에 대한 예측 및 이에 대한 조치를 위해서는 프로그래머의 고려를 필요로 하게된다.

프로그래머에 의해 결정된 Data Communication Network에 따라 구현이 완료된 프로그램들은 실제 현장에서 사용되고자 할때 사용되는 시스템의 하드웨어 및 소프트웨어 구성에 따라 수행 속도가 달라지게 된다. 물론 개발 시스템과 동일한 시스템에서는 정상적으로 수행되어 프로그래머가 원하는 수행 속도를 기대할 수 있으나 이러한 프로그램은 특수한 분야에서 특수 목적용으로 개발되는 프로그램들에 해당하는 사항이고 대부분의 프로그램들은 하드웨어 자원의 구성이 다소 차이가 있는 서로 다른 시스템에서 수행되어야 한다. 즉 프로그램의 portability가 제공되어야 한다. 이러한 portability를 제공하기 위해서는 기본적인 하드웨어 구성 및 시스템 소프트웨어의 제공 기능이 유사한 시스템에서는 정상적인 수행이 가능해야 한다. 그러나 병렬처리 소프트웨어들은 시스템의 하드웨어 자원의 구성 형태에 따라 처리 결과에 많은 차이가 있는 관계로 다른 시스템에서 정상적으로 수행되기 위해서는 프로그램의 변경이 불가피하다.

또한 같은 프로그램이라 할지라도 실제로 많이 사용되는 데이타의 형태에 따라 프로그램 개발단계에서 결정된 Data Communication Network가 적절하지 않을 수도 있다. 즉 병렬처리되는 각 수행 모듈들중 특정 모듈의 수행이 빈번하거나 데이타의 통신 형태가 주로 사용되는 정보에 따라 많은 차이가 있을 수 있다. 이러한 경우에는 프로그램 개발시에 발생하지 않았던 프로그램 수행상의 bottle-neck이 있을 수 있으며 따라서 프로그램의 전체적인 Performance에 큰 영향을 미칠 수 있다.

본 보고서에서는 이러한 문제점을 해결하기 위한 방법의 일환으로 Adaptive Distribution Network Reconfiguration Model을 제안하고자 한다. 즉 프로그램의 개발단계에서 사용되는 sample 데이타를 이용하여 통신 overhead를 최소화 할 수 있는 하나의 Data Communication Network을 구성한 후에, 실제 현장에서 사용될 때는 일정 기간동안 각 Task들 사이의 통신 형태를 분석하여 Communication Overhead를 최소화 할 수 있는 Distribution Network을 재구성하여 프로세싱 노드의 할당시에 이용함으로써 시스템 개발자가 현장에서 사용되는 시스템의 구성상의 특성을 고려하지 않아도 스스로 Performance를 극대화할 수 있는 모델을 제공하고자 한다.

2장에서는 우선 처리 속도가 매우 중요한 고려 대상인 프로그램들중 대표적인 멀티미디어 데이타처리 프로그램들의 특성을 분석해 보고, 3장에서는 병렬처리 시스템의 구성에 따른 시스템 자원을 분석하고 4장에서는 분산, 병렬언어인 CDL(Component Distribution Language)에 대하여 설명하며, 5장에서는 프로그램의 분산, 병렬처리에 있어서 고려되어야 할 점과 문제점에 대하여 알아보기로 한다. 6장

에서는 본 연구의 타당성을 제시하기 위하여 분산, 병렬시스템에서의 검색 프로그램에 대하여 분석하였고, 7장에서는 앞에서 제시된 문제점들을 해결하기 위한 방법의 하나로써 Adaptive Distribution Network Reconfiguration Model에 대한 소개를 하기로 한다.

제 2 장 멀티미디어 데이타 처리 프로그램의 특성

여 백

제 2 장 멀티미디어 데이타 처리 프로그램의 특성

멀티미디어 데이타 처리 프로그램의 가장 큰 특징중의 하나는 video 데이타와 같은 정보처리라 할 수 있다. 멀티미디어 데이타 처리 프로그램의 한 예로서 Video Conferencing을 위한 프로그램을 들 수 있다. 상대방의 영상이 실시간으로 화면에 나타나며 음성이 전달되고 공통의 윈도우에 논의되고 있는 내용을 표시하는 동시에 자신의 자료를 상태에게 전달할 수 있는 프로그램이라 할 수 있다. 이러한 프로그램에 대한 분석을 통하여 멀티미디어 데이타의 특성에 대하여 고찰해보기로 한다.

제 1 절 프로그램의 Module화

위에서 언급된 Video Conferencing 프로그램의 경우를 고려해 보면 한 시스템에서 동작되는 프로그램들이 몇개의 개념적인 모듈들로 나뉠 수 있다. 카메라로부터 자신의 영상을 입력받아 상대방에게 보내주고 상대방의 영상을 화면에 출력하며 필요한 데이타를 저장하거나 저장되어 있는 정보를 로드하여 상대방에게 전송하는 등의 독립된 모듈들로 분리해 볼 수 있다. 이와 같이 분리된 각 모듈들 사이의 데이타 전송 관계를 그림-2.1에 나타내었다.

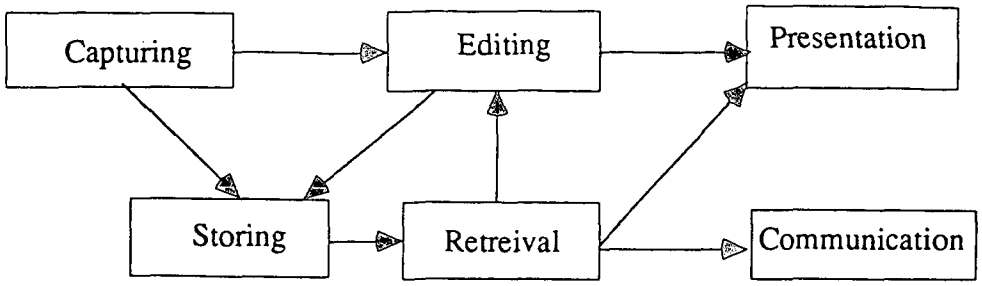


그림-2.1. 멀티미디어 데이타 처리를 위한 모듈들 사이의 관계

위의 그림에서 볼 수 있는 바와 같이 멀티미디어 데이타 처리 프로그램을 구성하는 모듈은 크게 6가지로 구분될 수 있다. 이들 각 모듈들은 일련의 데이타를 입력받아 정해진 처리 과정을 거친후에 다음 모듈에 전달해야 하며, 또한 각 모듈들에 입력되는 데이타는 연속적이며 따라서 각 모듈들은 빠른 시간내에 입력된 데이타를 처리하여 다음 모듈에 전달해야 한다. 즉 데이타의 처리 과정이 Pipeline 형태의 처리를 요하고 있으므로 단위시간내에 각 모듈들은 입력된 데이타에 대한 처리를 완료해야만 한다. 또한 위의 각 모듈들은 각 미디어마다 서로 다른 처리 모듈들로 구성되어야 하기 때문에 실제로는 많은 모듈들로 구성된다. 이와 같은 각 미디어 데이타에 대한 모듈들은 서로 동기화를 위한 통신이 필요하며 따라서 전체적인 모듈들 사이의 관계는 매우 복잡하게 된다.

제 2 절 대용량의 데이타 처리

멀티미디어 데이타 처리 프로그램이 처리해야 하는 데이타의 양은 기존의 문자처리 프로그램과는 비교가 되지 않을 정도로 많다. 단순히 500 x 500 의 해상도를 갖으며 256가지의 색상을 표현할 수 있는 모니터에 초당 30 Frame의 Video 데

이타의 출력을 위해서 처리되어야 하는 데이타 양을 계산해보면 초당 약 7.5M Byte의 데이타 처리를 필요로 한다. 즉 그림-2.1에서의 Capturing 모듈과 Presentation 모듈이 동시에 초당 7.5M Byte의 데이타를 처리해야 하는 것이다. 이렇게 방대한 양의 데이타 처리를 위해서는 많은 양의 Computation Power를 필요로 한다. 즉 Video Conferencing을 위한 프로그램에서는 많은 모듈들이 위와 같은 데이타를 동시에 처리할 수 있어야 하므로 훨씬 많은 Computation Power를 필요로 한다.

제 3 절 미디어들 사이의 동기화

멀티미디어 데이타에 대한 처리는 각각의 미디어별로 처리 루틴이 독립되어 있으며 시스템 외부로 부터 입력된 멀티미디어 데이타들은 각 미디어별로 분리되어 일련의 처리과정을 거친 후에 각각의 주변장치로 이동된다. 특히 비디오 데이타와 같은 경우는 음성, 영상, 텍스트 등이 통합되어 있는 대표적인 멀티미디어 데이타로서, 이들 데이타는 시간적인 동기가 이루어지지 않으면 멀티미디어 데이타로서의 역할을 하지 못하게 된다. 이러한 여러 미디어 데이타들 사이의 동기화를 위해서는 서로 다른 미디어 데이타를 처리하는 모듈들 사이의 빈번한 통신이 필요하며 이를 제어하기 위한 복잡한 과정이 요구된다. 또한 이러한 동기화는 단위 시간내에 이루어져야만 원하는 멀티미디어 정보가 정확히 출력될 수 있으며 정보의 의미를 잃지않게 된다.

제 4 절 실시간 처리

위에서 살표본 바와 같이 멀티미디어 정보가 제대로 처리되기 위해서는 각각의 수행 모듈들에서의 데이타 처리가 지정된 시간내에 이루어져야 하며 어느 한 부분에서라도 지정된 시간내에 처리되지 못하고 지연되면 멀티미디어 데이타의 의미를 일부 상실하게 되는 특성을 지니게 된다. 즉 멀티미디어 데이타 처리 프로그램은 실시간 처리를 해야만 하는 특성을 지니고 있다. 실시간 처리를 위해서는 단위 시간당 많은 양의 데이타를 처리할 수 있어야 하며 어느 한 모듈에서라도 프로세싱이 지연되지 않아야 한다. 따라서 멀티미디어 데이타 처리 프로그램의 정상적인 수행을 위해서는 많은 양의 Computation Power가 필요하다.

제 5 절 병렬처리 특성

실시간 처리를 위한 한 방법으로서 병렬처리 시스템을 사용할 수 있다. 그림-2.1에서 고려된 각 모듈들은 멀티미디어 데이타 처리를 위해 동시에 수행될 수 있는 특성을 지니고 있다. 한 예로 video Conferencing 프로그램의 경우 각 Site에서는 카메라로 부터 입력되는 Moving Image를 받아서 (Capturing Module) 상대방에게 네트워크를 통해 전송해야 하며 (Communication Module) 상대로부터 전송된 데이타를 받아서 화면에 출력해야 (Presentaion Module) 한다. 또한 동시에 임의의 자료를 상대방에게 전송해야 하는 경우 저장장치로 부터 데이타를 로드(Retrieve Module)해야 하며 전송된 자료중 필요한 것은 저장되어야 (Storing Module) 한다. 또한 회의 대상자들의 공통 윈도우에 대한 처리가 이루어져야 하는데, 이들 각 모듈들은 뚜렷하게 자신이 수행해야 하는 업무가 있으며 서로 독립되어 있음을 쉽게 알 수 있다.

위와 같은 각 모듈들의 병렬성을 고려할때 각각의 모듈이 하나의 프로세싱 노

드에서 수행될 수 있다면 하나의 프로세싱 노드에서 여러 모듈들이 단위 시간당 수행해야하는 일을 처리하기 위해 필요한 프로세스 switching 작업이[Shivar] 불필요하며, 특정 모듈의 수행 지연으로 인한 프로그램의 비정상적인 동작에 의한 정보 손실을 막을 수 있다.

여 백

제 3 장 분산, 병렬처리 시스템

여 백

제 3 장 분산, 병렬처리 시스템

여러 형태의 병렬처리 시스템들 가운데 최근에 많이 사용되고 있는 시스템 Architecture로서 Distributed Memory Architecure를 들 수 있다. 각 프로세싱 노드들은 공유의 메모리를 소유하고 있으며 다른 프로세싱 노드와의 데이타 공유를 위해서는 각 노드에 연결되어 있는 link를 이용하여 데이타를 주고 받아야 한다. 보통 이들 link의 수는 제한이 되어있으며, 직접 연결되어 있지 않은 노드와의 통신을 위해서는 중간 프로세싱 노드의 도움이 필요하다.

위와 같은 Distributed Memory Architecture를 구성하기 위해 주로 사용되는 프로세싱 노드중 대표적인 것으로 Transputer를 꼽을 수 있다. 1983년에 Transputer의 개발과[Barron, INMOS, Stevens] 더불어 Distributed Memory 형태의 Multiprocessor 시스템을 쉽게 구현할 수 있게 되었다. Transputer 의 주요 특성은 다음과 같은 몇개의 항목으로 대변될 수 있다.

- o RISC Architecture
- o 하나의 칩에 컴퓨터 주변회로를 Integrate 시킴
- o Built-in Multitasking Kernel
- o 2가지의 프로세서 우선순위
- o link를 통한 간단한 interconnection

프로세싱 노드로서 Transputer를 사용하는 멀티프로세서 시스템에서는 프로세서사이의 통신이 Link를 통해 이루어 진다. 이 Link는 양방향의 Serial Communication Channel로 초당 20Mbit의 전송이 가능하다. 각 프로세싱 노드(T800

의 경우)는 4개의 Link를 갖고 있으며 이들 4개의 노드에 의해 다른 노드와의 통신이 가능하다. 또한 각각의 Transputer에는 Local Memory가 존재하며 각 노드는 여러개의 Task를 동시에 수행할 수 있다. 조금 더 자세한 내용은 후에 설명하기로 한다.

제 1 절 분산, 병렬처리 시스템의 기본 개념

멀티프로세서 시스템에서 사용되는 시스템 소프트웨어(운영체제)중 Stand-Alone 시스템을 구축할 수 있는 운영체제는 대표적인 것으로 Helios를 들 수 있다 [Helios]. Helios는 Server-Client Model을 기반으로 각각의 Transputer 노드에는 가장 기본적으로 필요한 Kernel 및 loader등이 탑재되며 디스크 입출력과 같은 특수한 기능을 담당하는 서버는 필요에 따라 일부의 노드에서만 수행되게 된다.

각 프로세싱 노드는 자신이 속한 그룹의 프로세싱 노드들에 대한 연결 경로를 나타내는 Table을 갖고 있으며, 한 프로세싱 노드에서 다른 프로세싱 노드로의 데이타 전송을 위해서는 Table에 지정된 Link를 통해 데이타를 전송하게 된다. 만일 대상 노드가 직접 연결되어 있지 않을 때에는 중간 노드에게 데이타가 전달되고 중간 노드는 자신의 Path-Table에서 목적 노드로 도달하기 위한 Link를 선정한 후에 전달된 데이타를 원하는 목적지로 보내게 된다. 이러한 Link를 통한 데이타 및 신호의 전달은 2가지의 프로세서 우선 순위중 상위의 우선순위로 수행되며 중간에 forwarding 역할을 하는 노드는 그동안 자신이 수행하던 작업을 중지하여야 한다. Application 프로그램에서 각 Task 사이의 데이타 전송은 위와 같은 방법으로 어느 프로세싱 노드와도 통신이 가능하나 중간 노드가 있는 경우는 전체적인 면

에서 통신 Overhead로 인하여 성능이 저하되게 된다.

여러개의 Task로 구성된 하나의 Program이 병렬처리 되기위한 프로세싱 노드 할당은 load-balancing 관점에서, 수행되는 task가 없는 노드를 찾아 우선적으로 할당해 주고 모든 노드가 수행중인 task가 있을 경우는 가장 task가 적은 노드를 할당해 주고 있다. 즉 프로그램의 구현시에 고려된 Network Topology는 프로세서 할당시에 전혀 고려되지 않고 단순히 시스템 차원에서의 load-balancing만이 고려되고 있다.

1. 링크 Bandwidth

각각의 프로세싱 노드에 존재하는 4개의 Link Bandwidth는 초당 20 Mbits를 전송할 수 있으나 실제 데이타의 전송율은 14.5 Mbit / s 이다. 이것은 1 Byte의 데이타 전송을 위해서는 11 bit가 필요하기 때문이다. 그러나 실제로 사용자가 프로그램 구현시에 호출할 수 있는 통신 함수들은 일련의 통신 프로토콜에 대한 처리를 하기 때문에 프로그래머의 입장에서 위의 전송율은 더욱 떨어지게 된다.

2. 네트웍 구성

수십개의 Processing Node로 구성된 시스템에서는 대부분 각각의 노드가 소유하고 있는 Link를 임의의 Node의 한 Link와 연결이 가능하도록 하는 Switching 모듈이 존재한다. 이 모듈은 Network Configuration Manager와 같은 소프트웨어 모듈을 가지고 있으며 이 소프트웨어에 의해 Connection의 변경이 가능하다. 따라서 일부 운영체제에서는 시스템 사용자가 간단한 명령어의 수행에 의해 이를 변경할

수 있도록 하고있다.

3. 처리기 사이의 통신

하나의 Application 프로그램이 병렬처리 되기 위해서, 각각의 수행 단위 (Task 혹은 Process)들은 독립적인 코드 및 데이타 영역을 갖게 된다. 따라서 서로 공유하는 데이타가 2개 이상의 프로세싱 노드에서 동시에 처리되어야 하는 경우에는 해당 데이타를 실제로 소유하고 있는 노드로 부터 데이타를 복사해야 하고 데이타의 변경시는 복사하여 사용하고 있는 모든 프로세싱 노드에 이를 알려주어야 한다. 즉 각 프로세싱 노들 사이의 통신이 매우 빈번하게 이루어지게 된다.

4. 입출력 장치

Transputer 시스템에서 사용하는 주요 입출력 장치로는 하드디스크 및 사용자를 위한 Presentation 장비(예: CRT등)가 있으며, 이들 장치에 대한 인터페이스는 I/O Server에 의해 관리된다. 즉 하나의 Transputer 노드에 해당 장치에 대한 Server를 수행시키고 Network 상의 임의의 노드로 부터 I/O Request가 있을 경우는 Server에게 전달되어 처리된다. 따라서 멀티미디어 데이타 처리 프로그램과 같은 경우는 I/O Server가 수행되고 있는 노드에 Performance-Bottleneck 현상이 발생할 수 있다. 그러나 이러한 현상은 프로세싱 노드의 처리 속도보다는 입출력 장치의 속도가 현저하게 느리거나 Link Bandwidth 이상의 데이타 전송이 요구되는 경우에 발생하게 된다.

제 2 절 병렬처리 시스템 하드웨어

본 연구에서 사용된 병렬처리 시스템은 고도의 병렬 알고리즘을 실현시키는 message passing MIMD multicomputer이다. 이 시스템의 하드웨어의 구조를 그림-3.1과 같이 나타내었다.

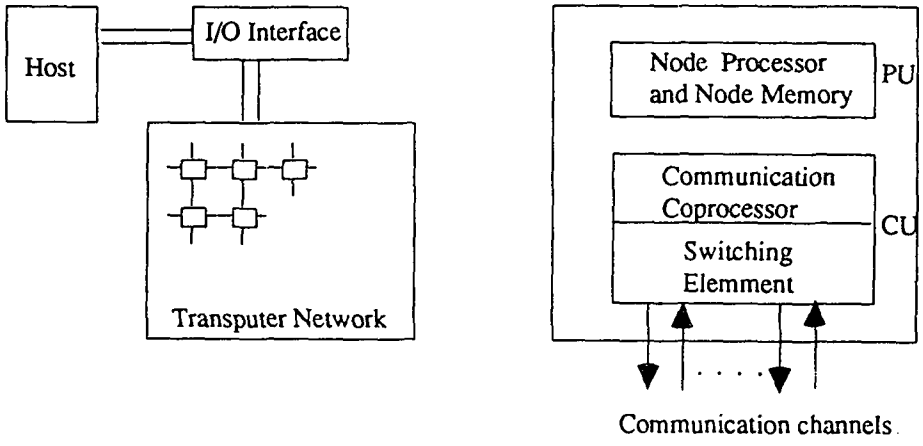


그림-3.1. 병렬처리 시스템의 하드웨어 구성

호스트 컴퓨터는 시스템 소프트웨어와 프로그램의 실행환경을 제공해 주는 I/O Interface를 통하여 이 시스템의 핵심인 병렬처리 시스템으로 연결된다. 이 병렬처리 시스템은 다수의 작은 컴퓨터들이 서로 메시지를 교환하며 병렬로 실행되는 병렬처리 컴퓨터이며 이 작은 컴퓨터를 노드라고 부른다. 노드 컴퓨터는 연산과 심볼릭 처리를 담당하는 PU(Processing Unit)와 다른 노드들 사이의 통신을 담당하

는 CU(Communication Unit)로 구성된다. 각 노드들은 다른 노드들과 통신을 위한 routing을 행하는 고속의 스위칭 소자를 사용하여 Interconnection Network를 구성한다.

1. 메시지 전달 방식 다중 처리기

본 연구에서의 병렬처리 컴퓨터는 프로세스들 간의 통신을 위하여 공유의 변수(shared variable)을 사용하지 않고 메시지 전달 방식 (message passing mechanism)을 사용하는 MIMD 컴퓨터이다. 메시지 전달 방식의 컴퓨터 구조에서는 프로세스와 메모리 사의에 switching network이 없는 점이 공유 메모리 방식과 다르다.

이 메시지 전달 방식은 프로세스와 메모리 사이의 통신과 프로세스간의 통신을 분리하여 생각할 수 있다. 따라서 소위 Von Neumann Bottlleneck이라고 불리우는 연산 프로세서와 랜덤 엑스 메모리(random-access storage) 사이의 통신에서 프로세서와 메모리 가 매우 밀접하게 결합될 수 있기 때문에 매우 작은 latency를 갖는다. 또한 노드의 기술 정도나 복잡한 정도에 따라 프로세스와 메모리를 함께 단일 칩이나 하나의 보드에 구현이 가능하다.

공유 메모리 방식은 쉽게 메시지 전달 방식의 특징들을 시뮬레이션 할 수 있고 메시지 전달 방식의 컴퓨터보다 코드나 데이타를 효율적으로 공유할 수 있으므로 보다 다양한 면이 있다. 그러나 이 두 가지 구조에서 공유 메모리 방식은 20개 미만의 프로세서를 갖는 시스템에 적합한 반면 메시지 전달 방식은 수 십개 내지 수 백개의 프로세싱 노드를 갖는 시스템에 적합하다.

2. Topology의 종류

병렬 처리 컴퓨터는 수 많은 프로세싱 노드들이 Interconnection Network를 통하여 통신을 한다. 이것들의 종류로는 Tree, Benes network, Shuffle Exchange network, Omega network, Indirect binary n-cube, direct binary n-cube와 같이 여러가지 많은 network topology들이 제안되거나 사용되어 오고 있다. 현재의 많은 컴퓨터는 k-ary n-cube 또는 k-ary n-cube와 동형인 mesh, indirect binary n-cube, Omega network를 많이 사용되고 있다.

3. 대기 시간

Network의 latency(대기시간)는 매우 중요한 성능 평가 요소이다. Network는 고도의 병렬성을 갖는 알고리즘을 지원하기 위하여 낮은 latency를 가져야한다. Network latency는 source에서 메시지의 첫 부분이 network로 들어가서 destination에서 그 꼬리가 나올때 까지의 시간을 말한다.

4. 처리 Unit (T800)

INMOS T800 트랜스퓨터는 64비트의 부동 소숫점 유닛을 가진 32 비트 마이크로 프로세서 이다. 고속의 프로세싱을 위하여 기본으로 2MBytes의 RAM을 내장하고 4개의 serial 통신 링크를 가지고 있다. 그리고 또한 고급언어를 지원하기 십도록 명령어를 갖추고 있고 OCCAM 모델을 직접 지원할 수 있도록 되어있다.

INMOS T800은 4GBytes의 어드레스 영역을 가지고 있으며 30MHz에서 동작할때 15MIP의 성능을 나타낸다. 그림-3.2는 T800의 내부구조를 나타낸다.

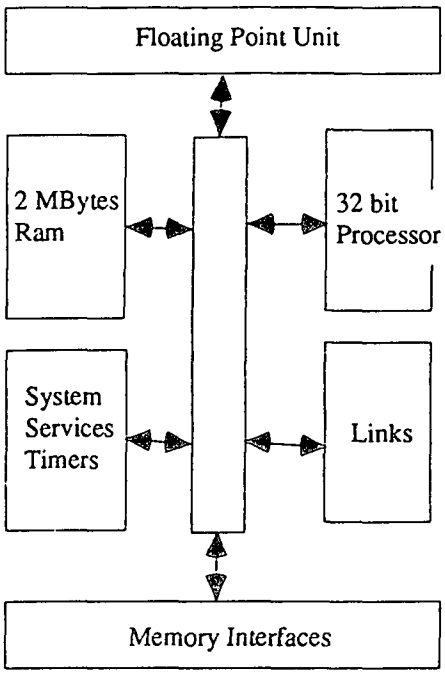


그림-3.2. Transputer T800 내부구조

5. 통신 Unit

CU(Communication Unit)는 노드와 노드의 메시지 교환을 고속으로 실현시키는 일을 한다. 서로 다른 노드에 있는 프로세스가 통신을 원할때 CU는 원하는 목적지로 메시지를 보내고 받는 모든일을 수행한다.

Network로 보내지는 메시지는 PU에서 만들어진 메시지의 첫 부분에 Routing tag를 붙여서 메시지 패킷을 만든다. Routing tag는 Interconnection Network상에서 source와 destination 사이의 상대적인 거리로 표시된다. 이 메시지 패킷은 패킷 스위칭에서 처럼 메시지를 일정한 단위로 나누어서 전송하지 않고 길이의 메시지를 그대로 전송한다.

Routing의 결정은 스위칭 소자의 하드웨어에 의하여 고속으로 행하여 진다. 메시지가 목적지에 도착하면 도착한 메시지는 스위칭 소자로 부터 인터페이스 회로를 거쳐 곧바로 버퍼에 저장되고 Communication Processor는 노드 메모리로 메시지를 전달한다.

CU는 스위칭 소자, 인터페이스 버퍼 및 coprocessor로 구성되며 각각의 기능은 다음과 같다.

- 스위칭 소자는 메시지의 routing을 결정하고 메시지 패킷을 원하는 노드까지 전달하는 역할을 하는 finite state machine이다.
- 인터페이스와 버퍼는 스위칭 소자와 communication coprocessor를 연결시키고 스위칭 소자에서 들어오는 데이타를 일시 저장하는 곳이다.
- Communication coprocessor는 노드 메모리와 스위칭 소자 사이의 데이타 아동을 제어하고 메시지의 destination을 상대 어드레스로 전환하여 메시지 routing tag를 만들어서 스위칭 소자에 전달하는 일과 메시지를 받을때 메시지 어드레스 테이블에 의해 적당한 노드 메모리로 전달하는 일을 수행한다.

6. 트랜스퓨터의 통신기능

INMOS의 트랜스퓨터는 병렬처리 CPU로서 현재 가장 많이 사용되고 있고 다른 CPU들과의 통신을 쉽게 하기 위하려 4개의 통신 링크를 가지고 있다. 그러나 4개의 통신 링크를 사용하여 다른 트랜스퓨터와 직접 연결하여 Network를 구성할 경우 인접하지 않은 트랜스퓨터와의 통신을 위해서는 중간 단계에 있는 트랜스퓨터의 중계가 필요하다. 대규모의 시스템을 트랜스퓨터의 링크에 의존하여 메시지를 전달할때는 많은 시간을 소요하게 되는데 이같은 메시지의 중계 부담을 덜기 위하여 INMOS에서는 각 트랜스퓨터들을 직접 연결할 수 있는 크로스바 스위치를 제공하고 있다.

제 3 절 분산, 병렬 운영체제 Helios

현재까지 수많은 병렬처리 시스템들이 개발되었고 실제 현장에서 사용되고 있다. 이들 병렬처리 시스템들중 특수 목적용이 아닌 범용 시스템 중에서 가장 많은 사람들이 관심을 갖고 있는 시스템은 Transputer를 이용한 시스템으로 현재 전자통신연구소 컴퓨터연구부에도 2대가 있다. 현재 보유하고 있는 시스템은 Parsytec 사에서 시스템을 구성한 Multicluster 2로서 16개의 Transputer 노드로 구성되어있는 시스템 Box와 Solbourne용의 VME 보드 (Transputer 노드 4개)로 설치되어있다.

현재 Multicluster 2는 Helios를 운영체제로 탑제하고 있으며, I/O (Disk) 서버는 Solbourne 시스템이 그 역할을 담당하고 있다. 본 절에서는 이 시스템을 사용하기 위해 기본적으로 필요한 Helios의 기본 개념에 대한 소개 및 주요 특징에 대

해 알아보기로 한다.

1. Helios의 기본 개념 및 내부 구조

Helios는 multiprocessor architecture용 운영체제로서 다음과 같은 운영체제로서의 특성이 있다[Helios].

- Distributed
- Multi-tasking
- Multi-Processor
- Multi-User
- Fault Tolerant
- Sympathetic to transputer architecture
- UNIX like User Interface

Host 시스템으로 가능한 것은 현재 SUN-3,4, IBM-PC등으로 인터페이스 보드가 있어야 한다.

Helios의 기본 개념은 Client-Server Model로서 컴퓨터 시스템의 통신 I/O, 등의 대표적인 기능들은 하나내지 두개의 Transputer가 Server 역할을 수행하며 다른 Client 노드(Transputer)에서 요청이 있을 때 Server에 Request Message를 전송하여 원하는 동작을 수행하도록 되어있다.

o Helios의 Shell과 Commands

UNIX의 C-Shell과 같은 Shell을 제공하고 있으며, PC의 MS-Windows와

Sunview의 I/O Server가 제공되고 있다. UNIX를 사용하던 사용자는 쉽게 사용할 수 있고, Multiprocessor Architecture의 기능을 위하여 Component Distribution Language(CDL)이 제공되고 있는데, 이것은 C-Shell 모드에서 flag를 setting 함으로써 모드를 변경할 수 있다. 이 CDL을 사용하여 command level에서 여러개의 command를 서로 다른 Transputer node에 분산하여 병렬 수행을 할 수 있으며 이때 시스템의 load balancer가 각 Transputer의 load를 분산하여 전체적인 시스템 성능을 향상시킬 수 있도록 하였다. CDL에 대한 자세한 내용은 5장에서 다루어 진다.

o Inside Helios

Helios의 내부 구성은 Client-Server Model을 기본 개념으로 하고 있으며, Device independent protocol을 제공하고 있다. 주요 특징으로는 다음과 같은 것이 있다.

- Client는 Server에게 message를 보냄으로써 원하는 resource를 access할 수 있다.
- Server와 Client는 Position에 관계없이 어느 곳에 있어도 무관하다.
- Modularity를 제공하고 있다.
- 사용자가 원하는 Server를 쉽게 확장 혹은 개발이 가능하다.

Helios의 가장 중요한 역할이라 할 수 있는 Processor Manager는 다음과 같은 역할을 담당하고 있다.

- Task creation and deletion
- Signal Delivery
- Environment enquiry

- Name table management
- Sever location map 관리

각 Transputer 노드마다 존재 가능한 것 중에 다른 하나는 Nucleus로서 다음과 같은 것들로 구성되어 있으며 크기는 약 50Kbytes 정도이다.

- Kernel : memory management

  Hardware-independent interface 제공

  Transparent message passing

  Process Creation and Scheduling

- System Library : Basic I/O 관련 함수들

  "System Call" interface

  General Server Protocol Interface

  Task heap management

  Resource Tracking (예: open streams)

  Environment enquiry

- Server Library : System Programming Server
- Utilities LIbrary : string copy등의 함수들
- Processor Manager
- Loader

각 Transputer 노드마다 기본적으로 가지고 있는것 중에 Loader라는 것이 있는데 이것은 해당 Transputer 노드에서 다음과 같은 역할을 수행하고 있다.

- Interprets load images

- Manages code sharing
- Loads and binds Resident Modules to programs

Helios에서 사용되는 Message Passing은 다음과 같은 특징을 갖고 있다.

- Messages sent to ports
- Location Independent
- Lightweight connections
- Delivery NOT guaranteed
- Timeout Feature

o Configuring your System

System의 구성을 위한 기본적인 정보 화일은 다음과 같은 것이 있다.

- host.con
- initrc
- passwd / motd
- default.rm / default.map

"host.con" 화일은 I/O Server를 위한 정보 화일로써 연결된 host의 종류 등의 구성 상태를 기술하는 화일이다. "initrc" 화일은 시스템이 boot-up 될때 필요로하는 과정이 기술된 화일이며, "default.rm" 혹은 "default.map" 화일은 Transputer 각 노드들의 physical connection을 기술하는 화일로 booting 과정에서 이용된다. booting이 완료된 상태에서 일반 user가 login시에 이용되는 화일이 "passwd" 및 "motd" 화일이다.

2. Helios에서의 병렬 처리 프로그래밍

Helios에서의 병렬처리 크게 두가지 방법이 있다. C 언어에 있는 병렬처리 statement를 이용하거나 혹은 Command Interpreter의 CDL mode에서 일련의 프로세스를 분산 병렬처리가 가능하도록 할 수 있다. 예를 들어 아래와 같은 명령어에 대해 생각해보면,

```
% cc file.c | asm -p -o file.o
```

"cc" 명령어와 "asm" 명령어 및 현재 수행되고 있는 Command Interpreter(shell)의 세가지 Command object는 현재 shell이 수행되고 있는 Transputer내에서 모두 수행된다. 이 명령어중에 "asm"을 다른 Transputer node 상에서 수행하려면 아래와 같은 명령어를 사용한다.

```
% cc file.c | remote 01 asm -p -o file.o
```

즉 shell과 "cc" 명령어는 같은 Transputer node에서 수행되고, "asm"은 node-ID가 01인 Transputer에서 수행된다.

o 프로세스와 Task

프로세스는 Transputer의 각 노드에서 독자적으로 수행될 수 있는 가장 기본적인 단위로서 하드웨어에 의해 직접 수행이 가능한 오브젝트이다. 즉 각 Transputer의 동작은 이들 프로세스를 수행하고 끝내는 일을 반복하는 것이다. Task는 이들 프로세서의 집합으로서 일련의 작업을 위한 프로그램 전체를 말한다. Task는 다음과 같은 성질을 갖는다.

- Owner of resources(memory, ports)
- Protection domain

- Fixed to one processor for lifetime

- May contain many processes

하나의 Task는 수행 코드뿐만 아니라 필요로하는 컴퓨터 시스템의 자원들을 총칭하는 것으로 오브젝트 코드의 protection 관리도 이 단위에서 수행된다. 또한 하나의 Task는 여러개의 process들로 구성될 수 있으며 이들 process들은 서로 다른 processor에서 수행될 수 있으므로 이들을 관리하기 위한 테이블 및 제어 데이타들도 Task의 일부라 할 수 있다. 비록 서로 다른 processor에서 여러 process가 수행되더라도 이를 총괄하는 것은 하나의 processor에서 담당하고 있으며 이 담당 프로세스는 Task가 완료될 때 까지 하나의 processor에서 수행된다.

o 프로세스의 생성

프로세스의 생성은 프로그램 코딩시에 아래와 같은 형태로 지정됨으로서 가능하다.

Fork(stacksize, function, argsize, ...);

지정된 'function'의 호출은 새로운 프로세스를 생성하게 되며, Task의 global 데이타를 공유하고, Semaphore나 Channel을 이용하여 동기를 맞춘다. 이렇게 생성된 프로세스는 return에 의해 종료되고 소멸된다.

o Task의 생성

새로운 Task의 생성은 프로그램의 코딩시에 다음과 같은 형태로 정의될 수 있다.

.....

if (vfork() == 0)

```
{
        execl(program, args.... , 0);
      _exit(1);   }
 ..............
 wait(&status);
  .....
```

위와 같은 방법으로 생성된 Task들 사이의 통신은 크게 4가지 방법으로 가능하다.이들 각각의 장단점을 간단히 요약하면 다음과 같다.

- Link를 이용하는 방법
  * 장점 : very efficient, flow control, reliable
  * 단점 : neighbour only, unmultiplexed.RE
- Messages를 이용하는 방법
  * 장점 : efficient, point-to-point, multiplexed
  * 단점 : unreliable, unbuffered, complex, no flow control
- Pipes를 이용하는 방법
  * 장점 : reliable, point-to-point, stream interface, flow control
  * 단점 : unbuffered, some overhead.
- Fifo를 이용하는 방법
  * 장점 : reliable, stream interface, buffered, flow control
  * 단점 : indirect, high overhead.

3. Helios의 개선점

현재 Helios가 탑제된 Transputer 시스템들은 대부분은 processing power를필요로하는 응용분야에 많이 이용되고 있다. 즉 inter-process, inter-task communication 양이 아주 적은 수치계산등의 분야에 아주 적절한 시스템이라 할 수 있다. 그러나 멀티미디어 데이타를 처리하는 Task와 같은 분야에는 각 process 혹은 Task간의 통신 양이 많은 관계로 병렬,분산 처리가 거의 효과를 볼 수 없는 상태이다. 즉 System Software중의 하나인 load-balancer의 역할이 미약하여 통신양이 많은 Task(혹은 프로세스)들에 대한 분산, 병렬처리가 무의미하다고 할 수 있다. 즉 다루는 데이타의 양이 큰 경우 이들 데이타의 통신을 조사하여 통신 overhead를 줄일 수 있는 방향으로 분산 병렬처리가 수행되도록 해야만 병렬처리의 효과를 얻을 수 있을 것으로 생각된다.

제 4 절 분산, 병렬 운영체제 TRACOS

1. TRACOS의 필요성

병렬화를 요구하는 응용프로그램들 즉, image processing 이나 수치해석 등을 효과적으로 처리하기 위하여 transputer가 많이 사용되고 있다. 이러한 부류의 application에서는 CPU-bound job이 높은 workload로 작용하지만 I/O-bound job 즉, 많은 data들이 Transputer-Networks상의 node간을 높은 rate을 가지면서 transfer 되는 것 자체도 각각의 node 에서의 CPU availability를 떨어뜨리는 workload로 작용한다.

본 절에서는 I/O bound job 및 communication job 등이 주요 workload인

Transputer-Networks 상에서 각 node간의 data transfer를 원활하게 하여주는 TRACOS(The packet-oriented Communication System)에 대하여 알아 보기로 한다. TRACOS는 특히 image processing 처리를 위한 system을 위하여 제작되었고 이미 상품화 되어있는 PERIHELION사의 Transputer Machine의 General Purpose Operating System인 HELIOS내에서의 communicatin overhead를 줄이기 위하는 것을 목적으로 하였다[Oehlrich].

어떠한 특정 병렬시스템의 성능을 최대화하기 위하여서는 그 시스템 내부의 dynamic한 상태라든가, 각각의 다른 node 간의 process 사이의 상호 작용을 monitoring 할 수 있어야 하는데, Transputer-Networks와 같은 병렬시스템 내에서 이러한 감시방법중에 가장 많이 호평받고 있는 것이 "Event-Driven Monitoring" 방식이다. Event-driven monitoring 방식은 monitor되어지는 시스템에서 순차적으로 발생되는 event를 감시하고 이 모든 event들을 "event-traces" 식으로 저장하기 때문에 programmer들에게 특정 program의 실행중 현 상태에 대한 충분한 정보를 제공한다. Transputer-Networks상에서 일어나는 software event를 monitor하는 방법은 2가지로 나뉘어질 수 있다. 하나는 감시되어지는 시스템내의 자체 clock을 이용하는 software monitoring 방식이고, 다른 하나는 시스템 내부가 아닌 외부 hardware clock를 이용하는 hardware monitoring 방식이다. Transputer-Network와 같이 여러개의 processing node들로 구성된 시스템에서, packet transfer time과 같은 각 node 간의 통신시간을 측정하기 위해 global time base가 필요하기 때문에 hardware monitoring 방식이 적합하다.

2. Transputer-Networks 상의 통신체제

1983년 부터 개발되기 시작한 transputer의 출현으로 shared memory 형태의 시스템에서 distributed memory 형태의 Multiprocessor system들이 대두하기 시작했다. Processing node를 transputer로 사용할때, transputer 간의 communication은link를 통하여서만 행하여진다. Link란 20Mbits/sec의 속도로 동작되는 양방향 bitserial communication channel이다. Link에 access하기 위하여 kernel은 "channel"이라는 data type과 channel 내에서 동작되는 IN과 OUT이라는 명령어를 제공한다. Channel의 사용은 link를 통한 process 간의 communication에 아무런 제한이 없을뿐더러 한 processor(transputer) 에서의 process간 communication에서도 사용할 수 있다. 그러므로 channel은 Transputer내 memory의 특정 variable로 구현되어 있고 이 channel variable은 link와 함께 미리 지정된 memory address로 access 되어진다.

Transputer의 통신방법인 channel의 또다른 특징은 "rendez-vous" 방식의 process coordination이다. 만약에 한 프로세스가(sender) OUT 명령어를 사용하면서 channel을 통하여 다른 process(receiver)에게 data를 보낼때, 이러한 행위는 receiver가 같은 channel상에 IN 명령어를 실행하여 data를 읽기 전까지는 sender를 봉쇄(block) 한다. 이와 마찬가지로 receiver가 sender의 OUT이 있기전에 미리 IN을 하였을때 receiver는 sender의 OUT이 있기까지는 block 되어진다. 이러한 단점을 보안하는 방법으로 queueing 방식을 사용할 수 있는데 이것을 hardware와 software로 모두 구현한것이 TRACOS System이다.

TRACOS : TRansputer COmmunication System

TRACOS System의 주된 목적은 Transputer-Network내에서의 한 sender가 다른

receiver로 data를 보낼때 이를 packet 형태로 routing 하여 주는 것이다.

각각의 transputer node에는 link를 통하여 packet이 들어올때 처리되는 절차가 있다. Receiver는 link를 통하여 들어온 data내에서 최종 destination 의 identifier를 fetch하여 최종 destination이 자기 자신이 아닐 경우에는 또 다른 link로 연결되어 져 있는 transputer node로 forward하여준다. 이러한 procedure는 packet이 최종 destination을 찾을때 까지 각각의 transputer node에서 행하여진다. 그림-3.3은 Transputer-Network 상의 각 transputer node 마다 있는 packet-orineted system인 TRACOS의 구조를 나타낸다. 각각의 4개의 priority로 구분되어 있는 input process 들은 4개의 associated 되어있는 link로 전해져올 packet을 기다린다. 물론 각각의 transpueter node 마다 transfer 되어올 packet을 buffering 할수있는 frame들을 소유하고있다. 그리고 routing 되어져야 할 packet이 link in을 통하여 들어왔을 때 이 packet은 associated 되어진 priority와 이 packet이 forward 되어져야 할 node에 대한 정보를 가지고 있는 routing table에 의하여 결정되어진 output queue에 넣어진다. 그 다음 각각의 output process들은 그들 자신의 queue에 쌓여있는 packet을 output link로 내보낸다.

각 노드의 user process들의 통신을 처리하기 위하여 2개의 process가 또한 존재하는데 이것이 local input process와 local output process이다. Local input process는 위에서 제시된 input process와 같은 기능을 가지고 있는데 이 process는 user process로 부터 내부 channel을 통하여 전달될 packet을 기다린다. 이 packet은 위에서 설명된 것과 같은 기능으로 output queue에 쌓여진 후 user process로 보내어진다. 여기서 user process와 communication system간의 data transfer 시간을 단축시키

기 위하여 내부 channel에서는 packet 자체가 아닌 packet을 가르키는 pointer만 전달되어진다.

위에서 보았듯이 packet이 routing 되어질때 필요한 정보를 가지고 있는 routing table은 user process가 실행을 시작하기 전에 network booter에 의하여 network configuration을 측정하여 각 node로 이 configuration을 분산시켜준다. 이 routing 정보를 바탕으로 각 node들의 TRACOS 모듈들은 다른 node와의 path를 찾아내어 packet들을 routing하여준다.

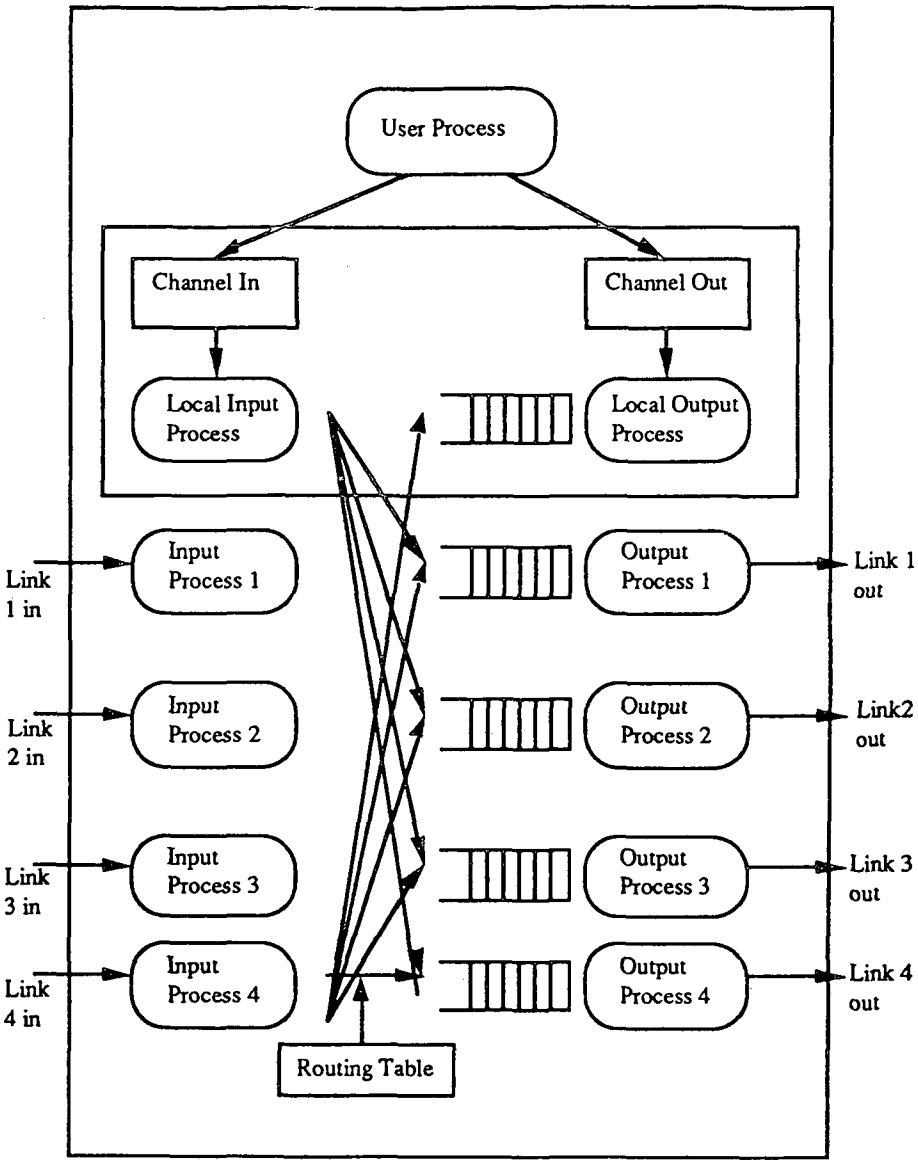


그림-3.3. TRACOS의 내부구조

3. Event-Driven 방식의 감시 장치

Event-Driven Monitoring은 events에 의하여 표현된 program activity의 dynamic한 상태를 나타낼 수 있다. 하나의 event란, program내의 특정 point나, hardware monitor에 의하여 탐지되어지는 processor bus 상의 어떠한 특정값에 의하여 정의할 수 있다. Hardware monitoring 방식을 사용함에 있어서, event의 정의나 인지는 힘든 작업이지만 software나 hybrid monitoring 방식에서는 event의 정의는 program 내에 측정에 필요한 명령어를 삽입시키면 쉽게 행하여 질 수 있다. 이런 측정을 목적으로 하는 명령어는 hardware monitoring(hybrid monitoring) 방식에 필요한 hardware system interface라든가 감시되어지는 system의 미리 지정된 memory 내부에 "event token"을 삽입시킨다. 이러한 측정 목적 명령어 삽입방식을 "Instrumentation"이라 부른다. 명백하게 "Instrumentation"은 측정의 목적과 성능평가(Performance Evaluation)의 취지를 지향한다.

Transputer-Network와 같은 분산시스템을 monitoring할 때는 일반적으로 사용되는 single monitor가 아닌 분산 monitor 시스템이 필요하다. Single monitor는 단지 한정된 숫자만큼의 event streams을 처리할 수 있기 때문에 Transputer-Network내에서 모든 process의 dynamic한 상태와 이들간의 interaction등의 인지 및 제어에 있어서 send/receive 방식의 communication events monitoring 방식은 적합하지 않다. 분산시스템 감시에 있어서 중요한 정보중에 하나가 "global time base"이다. 이는 분산시스템 내에서 일어날 수 있는 모든 events들의 정확한 시간을 측정하기 위하여

필수적인 parameter중의 하나이다.

TRACOS 시스템에서 hybrid monitoring 방식 즉 software와 hardware monitoring 방식이 결합된 것이 사용된다. 이 시스템에서는 monitor interface에 의하여 event-tokens이 시스템 내부에 삽입되어지고 hardware monitor에 의하여 record 되어진다. 이 방식을 위하여 TRACOS SYSTEM에서는 ZM4(global time base를 제공하는 분산 감시 시스템)을 개발하여 사용하였다.

가. 분산감시 시스템 ZM4

여기서는 ZM4의 hardware구조에 관하여 알아본다. ZM4는. 독일어의 약자로 Counter Monitor 4를 의미한다. ZM4는 다음과 같은 모듈로 구성되어저있다.

- CEC (Central Control and Evaluation Computer) : Master/Slave configuration 개념의 중앙처리 및 평가 컴퓨터.
- MA (Monitor Agent) :분산감시 agents
- Monitor Networks :Data channel과 tick channel.

CEC는 범용 Unix OS를 탑재한 mini-computer나 workstation이고 MA는 IBM-AT가 사용되었다. Data channel은 TCP/IP protocol의 Ethernet으로 CEC가 MA로 command를 보내고 setup 할때라든가 MA가 CEC로 측정된 data를 보낼때 사용된다. Event들은 DPU(Dedicated Probe Unit)에 의하여 인식되어지고 recording 되어진다. Tick channel은 각각의 DPU들의 local clock과 global clock인 MTG(Measure Tick Generator)와의 synchronization을 맞추기 위하여 사용된다. Monitor time base의

단위는 100ns이다.

이렇게하여 측정된 data들(event-traces)을 평가하기 위하여 "SAMPLE"이라는 tool이 사용되었고, Performance indice는 flow-oriented 평가에 적합한 Gantt-diagram(time-activity diagram)으로 표현하였다. Gantt-diagram은 특정시간 t에 대하여 병렬상태에서 지정된 activity들의 상호 의존도를 나타낼때 사용된다.

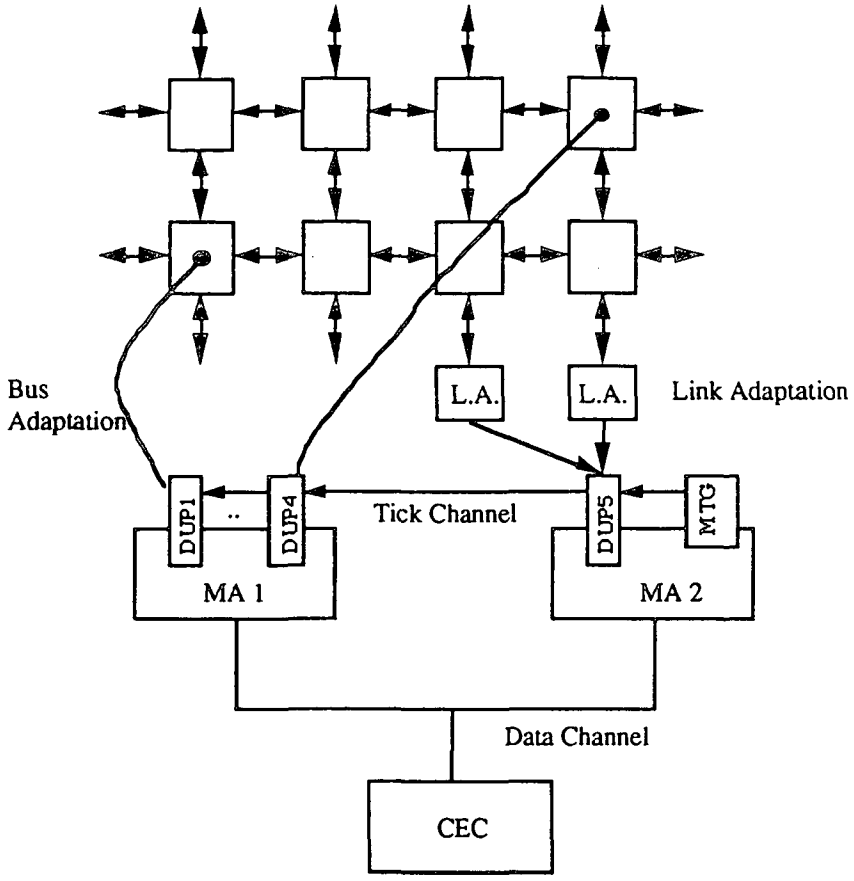


그림-3.4. ZM4 감시 시스템

나. Transputer-Networks에서의 hybrid 방식 감시 방법

앞에서 보았듯이 Event-Driven Monitoring 방식에서는 감시될 program들이 Instrumentation되어진다. Instrumentation은 monitor되어질 program의 세부 요소들을 "event" 단위로 처리한다. Transputer-Network 상에서 Hybrid-Monitoring 방식을 적용시키기 위하여 사용되는 방법에는 Bus Adaptation과 Link Adaptation등이있다.

- Bus Adaptation

이 적용방식에서는 event-token을 미리 지정된 memory에 assign한다. 이 memory를 Dedicated measurement Memory라 한다. Event 인지장치인 DPU는 이 특정 memory bus를 항상 감시하고 있어야 하며, 만약에 이 bus에 event-token이 주어지면 이것을 recording한다. 이 방식은 100ns(Monitor Time Base Interval)마다 하나의 측정명령어만을 사용하면서 적은 overhead로써 Transputer 내의 내부시스템상에 적은 영향만을 미친다. 하지만 단점으로는 Link Adaptation보다 hardware로 구현하기에 단가가 높다.

- Link Adaptation

이 방식에서 event-token은 link를 통하여 유출되어지고 DPU에 의하여 인지 및 recording 되어진다. 이것을 위하여 많이 사용되는 chip이 INMOS사의 Link Adaptor인 IMS C012이다. 이 방식은 Bus Adaptation 방식보다 감시되어지는 시스템 내부에 많은 overhead로 작용하면서 CPU availability를 떨어뜨린다(Communication Overhead). 각각의 측정 명령어는 시스템상에서 4 micro sec 정도의 overhead로 작용한다. 특히 Link Adaptation 방식의 단점은 감시되어지는 Transputer가 감시만을 목적으로 하는 Link를 소유하고 있을때만 가능하다.

4. TRACOS의 성능평가

TRACOS 시스템의 성능을 측정하기 위하여 여기서는 형태가 큰 Transputer-

network가 아닌 3개의 노드로 구성되어져 있는 작은 크기의 network를 고려하였다. 그 이유는 각각의 노드를 바탕으로 측정할 수 있는 종합적인 workload를 작은 network에서 조금더 세밀히 관찰할 수 있기 때문이다.

구성된 Network는 그림-3.5와 같다. T1은 Sender로써 Receiver인 T3로 2개의 구성요소를 (Packet size와 Packet rate) 보내는데 이는 T2를 통하여 이루어진다. T2 내에서는 W라는 통신에는 아무런 영향을 주지않는 Precess가 실행된다(Loop process 형태). 이 방법은 낮은 우선순위를 갖는 user precess에게 미치는 통신 load를 측정하기에 적합하다. 감시 시스템으로는 앞에서 설명되어진 ZM4 System(Global time base)으로써 각각의 노드에 Link adaptation 방식으로 설치되어 있다.

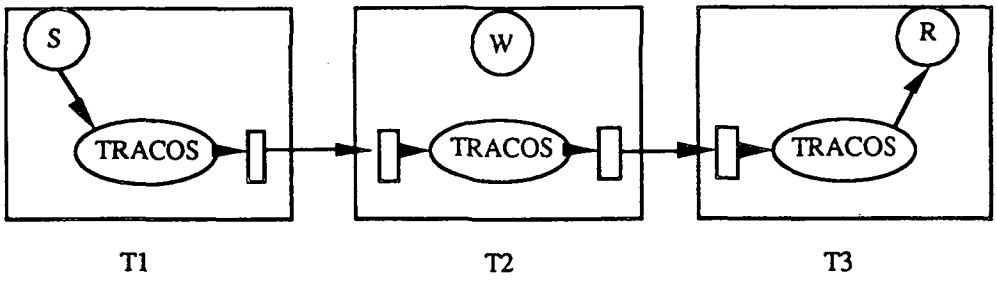


그림-3.5. 감시 Transputer-Network

가. 성능평가 지침

* 통신상태 (Flow-orinted Evaluation):

Sender와 Receiver 간의 통신상태를 Gantt diagram을 이용하여 이 2개의 노드간의 상호관계를 측정하였는데 그 결과는 다음과 같다. Transfer rate는 packet의 크기

에 따라 변화를 보였고 특히 어느 특정 한계점에 도달하면 Link bandwidth의 물리적으로 20Mbits/sec라는 한계가 있기 때문에 packet 전송시간은 TRACOS 시스템내의 Internal queue의 크기에 따라서 결정지워 지는 것으로 나타났다.

* TRACOS 시스템이 User process에 미치는 영향:

Packet 전송관리를 위한 통신 시스템에 의하여 사용되는 CPU time은 user process들을 위하여 최소화 하여야 하는데, 여기서는 TRACOS가 user process에 미치는 영향에 대하여 알아본다.

- 고정된 Packet size를 전송할때, Packet rate이 커지면 user process의 CPU availability가 떨어짐.
- 고정된 Packet rate으로, Packet size가 커지면 user process의 CPU availability가 떨어짐.
- Packet size와 Packet rate의 값이 고정되어 있을때, Packet의 길이가 길어질수록 CPU availability가 떨어짐.

이렇게 나타난 자료를 바탕으로 user process들에게 미치는 최악의 영향은 작은크기의 packet을 가능한 한 많이 전송할때 나타난다.

*TRACOS의 Packet rate와 전송시간 관계:

Packet size가 256 bytes보다 적을때 : 이때 Packet rate는 Link의 물리적인 한계점이 아닌 Packet 관리나 queue 관리등 Memory 할당에 사용되는 시간에 의하여 결정된다. 얻어진 자료로는 16 bytes의 Packet보다는 64 bytes의 크기를 갖는 Packet을 사용할때 더욱 효율적으로 전송이 되어졌다.

Packet size가 256 bytes보다 적을때 : 이때는, Link의 물리적인 한계점이 직접적으로 작용한다. Software적으로 이 한계점을 극복하기 위하여 사용할 수 있는 방법으로는 TRACOS 시스템에서 input process와 output process를 병렬화 하는 방법을 새각할 수 있겠지만 이것을 구현하기에는 여러가지 어려움이 있는데, 이는 후에 설명된 변형 TRACOS 시스템에서 언급한다.

나. TRACOS의 기능분석

Input process와 Output process를 그림-3.6과 같이 기능별로 구분하여 나타내어 보았다. 각각의 사각형들은 감시되어지는 points(instrumented event)들이다.

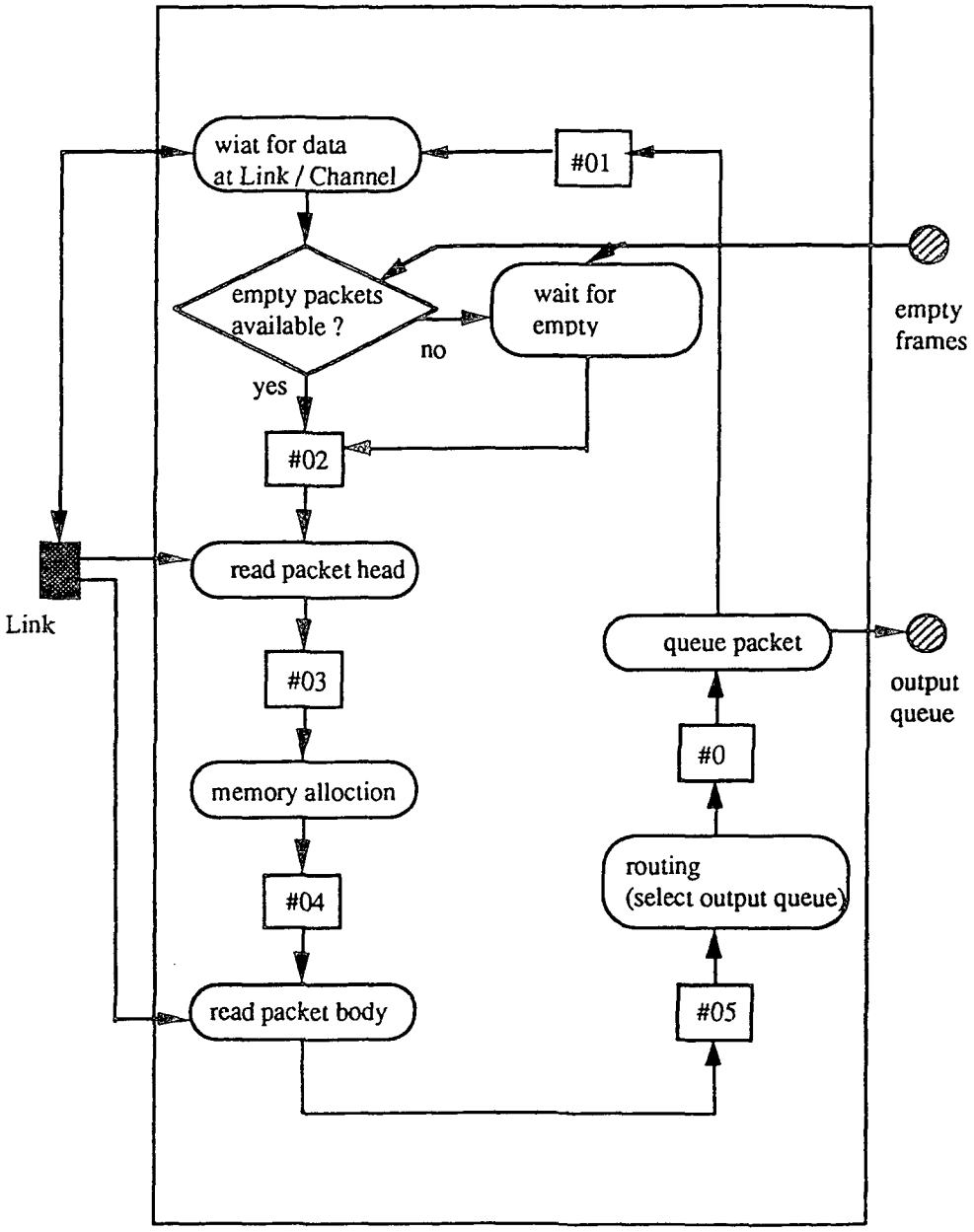


그림-3.6.a. TRACOS 의 input process

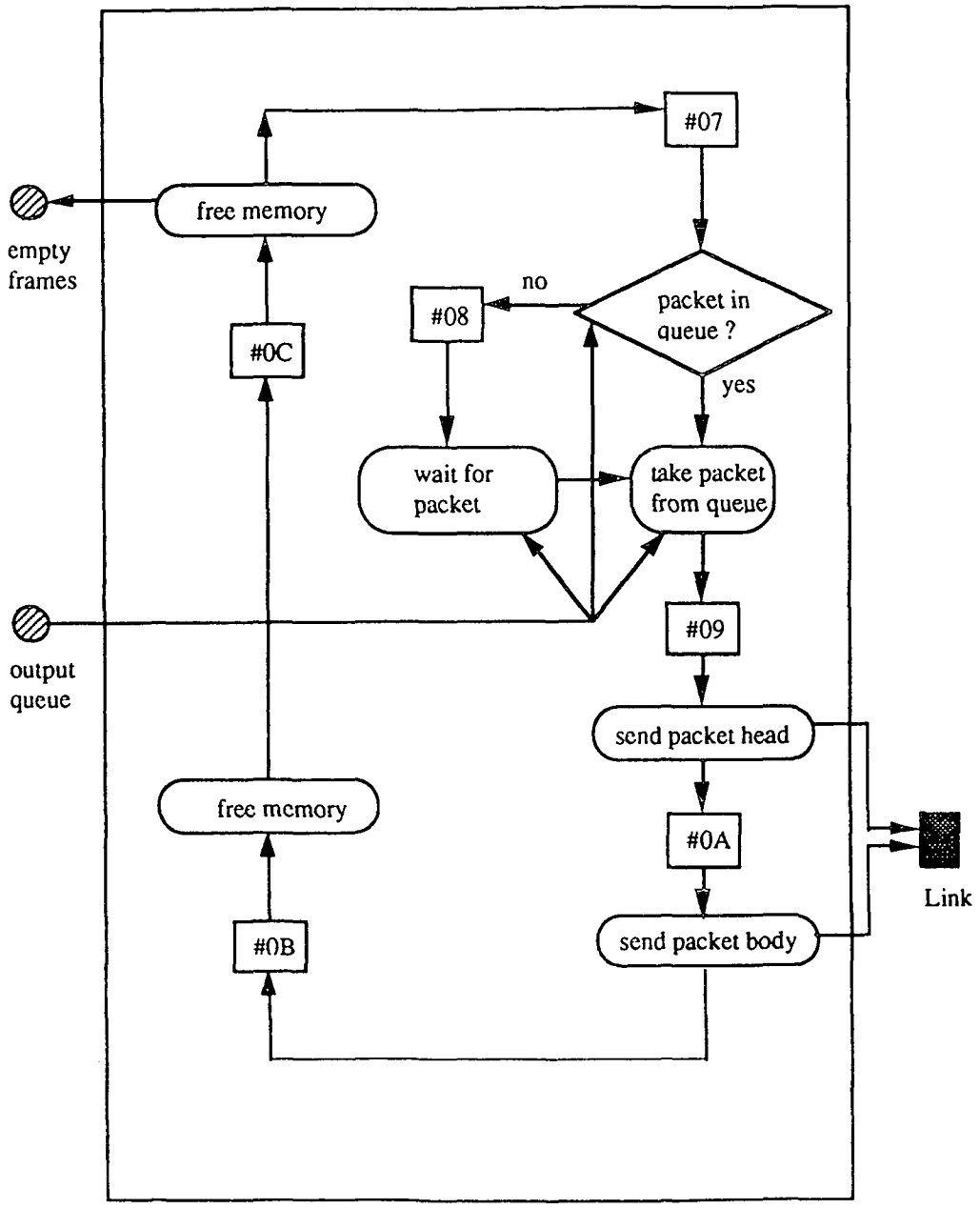


그림-3.6.b. TRACOS의 output process

*감시결과

T2에서 관찰된 결과. 비효율적인 Process scheduling (input process와 output process간의)으로 인하여 link in과 link out이 idle 상태로 자주 있었고 idle 상태에 있다가 한 순간에 과부하가 걸리는 것으로 나타났다. 이를 해결하기 위하여 Process scheduling을 바꾸는 방법도 있겠지만 이는 많은 노력을 요하기 때문에 여기서는 다른 방법을 사용하였다.

*변형 TRACOS 시스템

Output process를 2개의 서로 다른 작업을 하는 output process(O1)와 output process(O2)로 나누어 TRACOS 시스템을 재구성 하였다.(그림-3.7) O1은 packet의 header관리와 memory allocate/free 관련 작업만 하고 O2는 이 header만을 제외한 Packet을 전송하는 작업만 하면 되므로 O1이 memory 할당을 요구하면서 block 되어 지더라도 O2는 계속하여 packet을 전송할 수 있다.

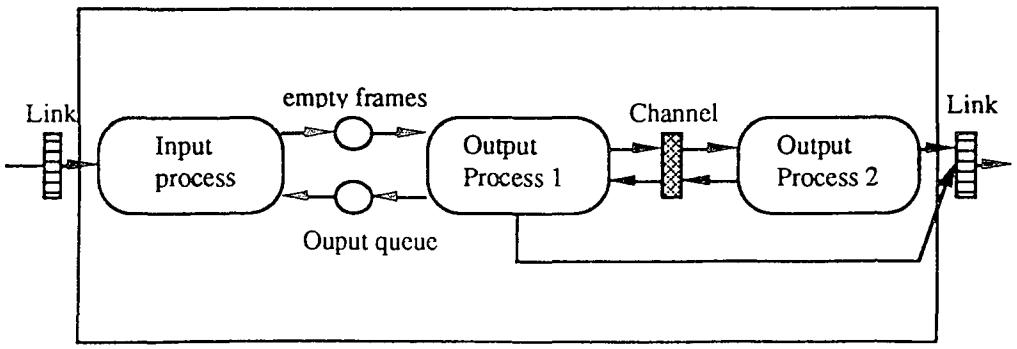


그림-3.7. 변형된 TRACOS의 processes

5. 개선점

본 장에서는 Transpuer-network에서의 Packet-oriented 통신시스템인 TRACOS에 대하여 알아보았다. 이것에 대한 분석은 3개의 transputer로 구성된 network를 hybrid-monitoring 방식과 instrumented된 종합적인 workload를 조사하여 이루어졌다. 또한, 이 network는 "event-driven" 방식의 하드웨어 감시 시스템인 ZM4에 의하여 측정되었고, TRACOS 그 자체는 소프트웨어 방식에 의하여 측정되었다.

감시결과, 비효율적인 input process와 output process들의 관리로 인하여 Link의 사용율이 저조하였기에, 하나의 output process를 packet 전송과 packet 관리(allocate/free)를 각각 전담하는 2개의 output process로 나누어 TRACOS를 변경 시켰는데, 그 결과 기존의 TRACOS 시스템보다 25%의 성능을 향상시켰다. 그리고 또한 TRACOS가 각 노드에서 user process에 미치는 영향에 대하여 조사해 본 결과 가장 효율적인 packet의 크기와 packet의 전송율은 각각 4Kbyte와 50 packet/s로 나타났다. 이 data로써 user process의 CPU time 사용율은 95%였다.

TRACOS 시스템에서는 Helios 시스템과 같은 Transputer machine을 위한 범용 시스템에서 각 노드간의 통신기능을 향상 시키기 위하여 Link buffering 기능과 routing 기능을 첨부하였다. 이것과 때를 맞추어 PERIHELION사 에서도 (Helios 제작사) 이 기능을 도입하여 buffering 기능과 routing 기능을 kernel 자체에 첨부시키는 작업중에 있고, 각 노드에서 사용할수 있는 buffer의 숫자와 packet의 크기 등을 user들이 사용하고자 하는 응용프로그램에 맞게 변경 사용할 수 있게 하고 있

다.

기존의 transputer 시스템의 사용은 computation power 만을 요구하는 프로그램에서 많이 사용되고 인정받아 왔지만, 앞으로는 multi-media나 natural language와 같은 대용량 데이타 처리를 위하여 많이 사용될 것이다. 대용량 데이타 처리를 위한 응용프로그램에서는 computation power 뿐만 아니라 현존하는 transputer 시스템에서는 하드웨어 한계점에 빠르게 도달할 것이다 (Link speed = 20Mbits/s). 그래서 지금 개발되고 있는 INMOS사의 T9000과 같은 경우에는 omputation unit와 communication unit이 하드웨어적으로 분리 되어져 있는 것은 물론이고 physical link의 숫자는 기존의 모델인 T800과 같이 4개이지만 논리적 link의 형태로 6개까지 사용 가능하다. 하지만 이렇게 한다해도, Link speed는 20 Mbits/s에 머무른다. 이 물리적인 한계점을 극복하기 위하여 현재 사용가능 한 기술로는 2가지가 있다. 첫번째로는 하드웨어 변형방법으로, 각 노드간의 Link를 bus 형태나 optical fiber를 사용하여 연결하는 방법인데, 이 방법은 시스템의 복잡도나 단가 때문에 구현하는데 많은 문제점이 있을것이다. 두번째로는 소프트웨어 변형방법인데, Interconnection Network 변형방법이다. 이것은 임의의 network 상에서 특정 응용프로그램의 실행중 얻어진 데이타를 바탕으로 이 특정 응용프로그램에 가장 효율적인 network를 찾아내어 기존의 network를 변형시키는 방법이다.

이 방법의 장점은 다음과 같다.

- 하드웨어 변경사항없음
- 어떠한 응용프로그램에도 적용가능
- 응용프로그램 실행중 어떠한 overhead로도 작용 하지 않음

제 4 장 분산, 병렬언어 CDL

여 백

제 4 장 분산, 병렬언어 CDL(Component Distribution Language)

본 장에서는 CDL(Component Distribution Language)을 이용하여 Helios 환경하에서 어떠한 방법으로 병렬 programming하는가에 관하여 기술한다[Cdl]. CDL의 기본 목적은 programmer가 한 application을 여러개의 program component(task 단위)로 정의하고 또한 이들 task 간의 Interconnection을 정의한 후 (이 정의 syntax가 CDL 임) Helios에게 transputer machine 내에서 할당받은 자원을 (transputer의 갯수) 바탕으로 이 program component들의 분산, 병렬 실행을 가능하게 하는 것이다.

제 1 절 CSP 모델

병렬 프로그램 모델에는 사용하고자 하는 hardware나 실행하고자 하는 application에 따라 그 종류가 다양하지만, 그 중에서도 CSP(Communication Sequential Processes)는 현재까지 가장 많이 사용되는 병렬 프로그램 모델중에 하나이다. 그리고 지금 기술하고자 하는 Helios CDL language, transputer 그리고 occam language들 모두 이모델을 바탕으로 제작된 것들이다. CSP의 기본 개념은 다음과 같다. 한 application 은 여러개의 실행 component(task)로 나뉘어 지고 이 각각의 task들은 소스로 부터 ("소스"라 함은 다른 task) data를 받아서 이것을 처리한 후 다른 소스에게 그 결과를 보낸다. 예를 들어 설명하면 그림-4.1 에서 보는 바와 같이 어떠한 application을 8 개의 box로 표현한다. 이 8 개의 box를

"sequential process"라 하자. 이 각각의 box들은 다른 box 로 부터 보내오는 data를 처리한 후 그 처리된 data를 다른 box로 전송한다. 이 시점에서 병렬화가 가능한 이유는 이 각각의 box들은 각각이 서로 다른 processor 내에서 (transputer network 내에서) 실행되기 때문이다. 그리고 box 간에 data를 주고받는 것은 communication channel을 통해서이다.

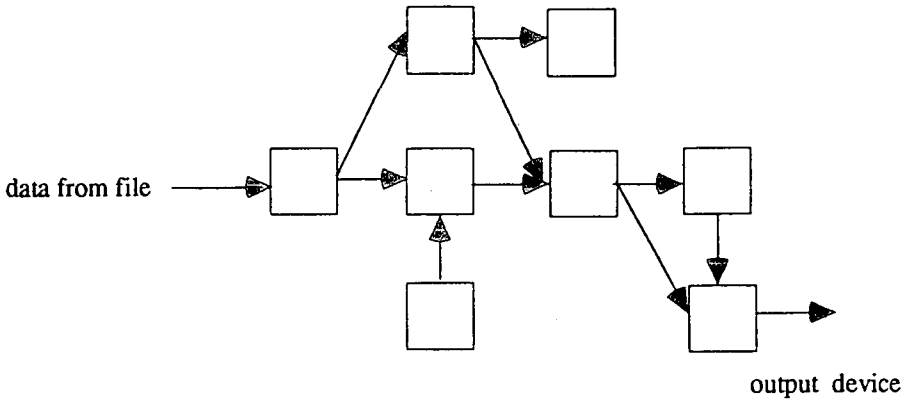


그림-4.1. CSP 모델

Occam language는 위에서 설명된 사항들을 low level에서 implement한 것이다. 즉 각각의 box들은 single Occam process들이고 이 box간의 communication channel들은 Occam channel (transputer link)인 것이다. 즉 각각의 사용자가 application을 실행하기 전에 explicit하게 사용하고자 하는 transputer의 갯수 및 그들간의 communication channel을 특정한 tool(예 : mtool 등) 을 사용하여 정확하고 확실하게 정의하여야만 한다. 이것은 사용자가 이용하고자 하는 transputer network 나 정의된 process간의 data들의 통신 상태에 대하여 사전에 정확한 정보를 가지고 있어야만 한다. 이러한 feature는 Occam language의 장점이자 단점인 것이다. 사

용자가 실행하고자 하는 application에 가장 적합한 환경을 구축해줌으로써 가장 optimal한 실행결과를 기대할 수 있지만 사용자가 만약에 많은 경험을 가지고 있지 않은 초보자일 경우에는 이러한 환경구축 작업이 overhead가 되기 때문이다. 이 단점을 보안한 것이 Helios의 CDL이다. 즉 CDL은 이러한 환경구축 작업을 high level에서 처리해 준다. CDL에서는 앞에서 언급된 box를 "task"라 하고 실행하고자 하는 application을 "task force"라 정의한다. 이 각각의 task들은 따로따로 작성되어지고 compile 후 debug 되어진 독립된 program들 이다. 또한 사용된 언어가 서로 다를 수 도 있다(C, Pascal 또는 Fortran). 사용언어가 다를 시에 주의해야할 사항은 각각의 task들간의 주고받는 data들의 type을 통일 시켜주어야 한다. 그리고 또한 서로 다른 성질의 network상에서 task force를 실행하고자 할때 (T800 & T414 network와 Intel사의 i860 network상에서)이 서로 다른 network간의 communication facility가 존재한다면 원하는 task force를 실행하는데는 아무런 문제가 없다.

서로 다른 task 간의 communication type은 Unix system의 pipe와 같은 style로써, 이 통신기능은 task force가 실행을 시작할 때 "Task Force Manager"에 의해 자동으로 만들어 진다. 이 통신 기능들은 Unix와 같은 style의 pipe를 사용하기 때문에 Unix system에서 사용하는 표준 I/O system call을 이용함으로써 병렬성을 지원하기 위하여 다른 형태의 언어를 첨가할 필요가 없다. 이와같이 CDL 언어의 기본 목적은 프로그래머가 실행하고자 하는 task force를 구성하는 각각의 task내의 component와 task간의 communication path(표준 Unix system 의 pipe style)를 정의하는 것이다.

제 2 절 CDL 언어

1. Task force 실행방법

기본적으로 특정 task force를 실행시킬 때 필요한 Helios Server 중의 하나가 "Task Force Manager"이다. Task Force Manager(이하 tfm)에 의하여 실행, 관리되어지는 task force들의 집합을 display하려면 다음과 같은 command에 의하여 행하여진다.

ls /tfm [CR]

*** 여기서 보는바와 같이 command "ls"는 Unix system에서 사용되는 것과 조금은 다른 성격의 command이다. 물론 Unix system에서와 같이 file description을 display할때도 사용되지만 위 예와 같이 현재 관리 되어지고 있는 task force 등이나 현재 trasputer network상의 각각의 processor들의 status 등을 display하는데에도 사용된다. ***

우선 task force 실행방법에 관하여 알아보기 전에 먼저 Helios 환경하의 실행 object code의 종류에 대해 알아본다. Helios하의 실행 object code에는 "program"과 "complie된 task force"가 있다. Program은 특정언어의 compiler(C, Pascal compiler 등)에 의해 만들어진 object code이고 Compile된 task force란 Helios shell 모드의 하나인 CDL 모드 내에서 CDL compiler에 의하여 형성된 object code를 의미한다. 그럼 CDL 모드란 무엇인가에 대하여 알아본다. Helios shell은 두가지 다른 모드를 가지고 있는데, 하나는 Unix shell과 같은 "Unix mode"이고 다른 하나는 "CDL

mode"이다. Unix 모드에서 Helios shell은 여러가지 Unix command와 실행 프로그램 등을 실행시킴으로써 표준 Unix C-shell과 같은 동일한 service를 제공한다. 그러나 이 모드에서는 compile된 task force는 실행시킬 수 없다. CDL 모드에서는 주어진 모든 명령어(즉, command, 실행 program 뿐만 아니라 compile된 task force까지)가 tfm에게 보내지는데 tfm은 보내어진 명령어를 현재 transputer network 상의 사용가능한 processor 들에게 분산, 병렬실행 시켜준다.

위에서 보는바와 같이 Helios shell의 Unix 모드와 CDL 모드와의 차이점은 다음과 같이 정리할 수 있다. Unix 모드에는 현재 사용가능한 processor의 숫자에 관계없이 모든 명령어들이 root processor내에서 sequential하게 실행 되어지고, CDL 모드에서는 모든 명령어들이 현재 transputer network상에서 골고루 분산, 병렬처리 되어진다.

Unix 모드에서 CDL 모드로 switching하는 방법은 "cdl"이라는 system variable을 지정함으로써 가능하다. 즉 다음과 같은 command를 실행시킴으로써 mode switching한다.

set cdl (Unix mode -> CDL mode)

unset cdl (CDL mode -> Unix mode)

*** Helios shell의 default mode는 Unix mode 임 ***

2. Task force 정의를 위한 parallel constructor

CDL 에서 사용할수 있는 parallel constructor 는 다음과 같이 4가지가 있다.

| , <> , ^^ , |||

| : name : pipe constructor

syntax : A | B

mean : A -> B (uni-directional pipe)

<> : name : subordinate constructor

syntax : A <> B

mean : -> A B <- (bi-directional pipe)

^^ : name : parallel constructor

syntax : A ^^ B

mean : A B (no communication)

||| : name : interleave constructor

syntax : A ||| B

mean : -> -> A lb B (automatic insertion of lb "load balancer") <- <-

이 4가지 parallel constructor의 우선순위(priority)는 다음과 같다.

^^ , ||| , | , <>

low ------> high

그리고 또한 이 parallel constructor 간의 combine이 가능하며 "left to right chain"에 의하여 표현된다. 다음의 예로써 parallel constructor 들로 표현할 수 있는 network를 그림-4.2로 나타내어 보았다.

예) A(<>B,<>C(|D))|E(<>(F<>G))|H (그림-4.2 참조)

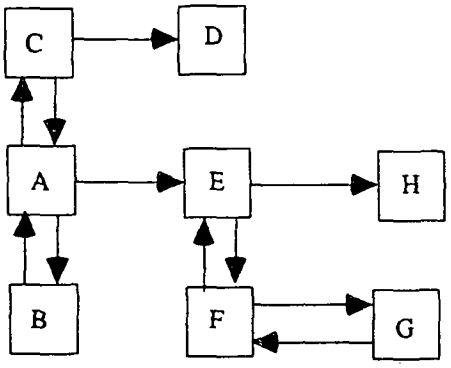


그림-4.2. Parallel Constructor 모델

CDL parallel constructor는 또한 표준 Unix system에서 제공하는 redirection mechanism 및 named pipe를 제공하기도 한다. 다음의 예는 redirection 및 named pipe를 이용하여 ring 형태의 task force를 표현한다.

예) (A <! loop) B (B>! loop) (그림-4.3 참조)

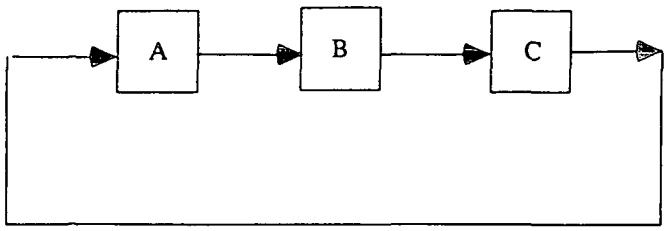


그림-4.3 Loop 모델

3. 스트림

여기서는 parallel constructor에 의해 형성된 task force 내의 communication channel인 stream이 어떠한 방법에 의해 할당되어지고 사용되는지에 대하여 몇가지의 task force를 예로 들면서 살펴본다.

예 1) A ! B

file descriptor	component A	component B
0	stdin	input from A
1	output for B	stdout
2	stderr	stderr

예 2) A <> B

file descriptor	component A	component B
0	stdin	input from A

1	stdout	output for A
2	stderr	stderr
3	unused	unused
4	input from B	unused
5	putput for B	unused

예 3) A ^^ B

file descriptor	component A	component B
0	stdin	stdin
1	stdout	stdout
2	stderr	stderr

예 4) A <> B <> C

file descriptor	component A	component B	component C
0	stdin	input from A	input from B
1	stdout	output for A	output to B
2	stderr	stderr	stderr
3	unused	unused	unused
4	input from B	input from C	unused
5	output for B	output for C	unused

예 5) A III C

이것은 A <> B <> C 와 같다. (B 는 load balancer)

4. Component 선언문

Task force를 정의하기 위하여 parallel constructor를 이용하여 CDL script를 만들듯이 component 선언문을 이용하여 각각의 task 들을 좀더 정확히 정의 할 수 있다. Component 선언문의 syntax는 다음과 같다.

COMPONENT component_name ([code;] [processor;] [puid;] [attrib;] [memory;]
[streams;] [...] [.....])

component_name : Task의 이름. 특정 compiler에 의하여 만들어진 object code.

code : Object code의 path 및 이름

processor : Task가 실행될 processor의 종류. (예; T800, T414, ANY)

puid : Trasnsputer network 상의 특정 processor의 logical name.

attrib : Resource map에 지정된 attrib를 사용. (Resource map : Helios 가 booting 할때 사용되는 transputer network 환경구축 configuration file)

memory : Memory size를 지정.

streams : 사용자가 explicit하게 streams를 지정.

이것 외에도 다른 정보를 지정할 수있는 field들이 있지만 가장 많이 사용되는 것들만 나열해보았다. 다음의 예는 한 CDL script 파일의 내용이다. 이파일은

task force 정의와 component 선언문을 포함한다.

```
#
# 이것은 comment 임
# component 선언문
#
component master { processor T800; memory 50000; }
component slave  { processor T414; memery 20000; }
component display { attrib frame_store; }
#
# task force 정의
#
master ( <> slave, <> slave, <> slave ) | display
```

5. 중복 명령어

Task force를 정의함에 있어서 같은 component를 여러번 똑같이 describe 한다는 것은 약간은 피곤한 작업이 될 것이다. 실제로 대부분 어떠한 task force은 보통 하나의 control program(master)과 이 control program에 의하여 구동되는 여러개의 같은 작업을 하는 worker program(slave)으로 구성되어진다. 이러한 task force를 정의하기 위하여 CDL 은 Replicator라는 facility를 제공한다. CDL에서 제공하는 replicator에는 2 가지 종류가 있다. Parallel constructor 앞에 위치하는 pre-replicator와 뒤에 위치하는 post-replicator가 있다. 이것들의 syntax는 다음과 같

다.

* pre-replicator

component1 [number of component2] parallel_constructor component2

(예) A [3]| B <=> A | B | B | B

A [3] ||| B <=> A <> lb 3 (<> B, <> B, <> B)

* post-replicator

parallel_constructor [number of component] component

(예) | [3] A <=> A | A | A

||| [3] A <=> lb 3 (<> A, <> A, <> A)

물론 post-replicator로 표현된 task force는 pre-replicator로도 표현이 가능하다.

Replicator는 interleave constructor와 같이 사용할 때 특별한 장점이 한가지 있다. Interleave constructor로 표현된 task force 내에서는 앞에서 설명된 바와 같이, master component는 단지 load balancer와만 서로 상호작용(interaction)을 하고 이와 마찬가지로 slave component들도 master component와는 직접적인 작용없이 load balancer와만 상호작용을 한다. 다시말해 이것은 원하는 task force를 slave component의 program source를 고치지 않으면서 단지 CDL script file만을 바꾸면서 slave component 의 숫자를 원하는대로 바꿀 수 있다는 것을 의미한다.

제 5 장 프로그램의 분산 병렬처리

여 백

제 5 장 프로그램의 분산 병렬 처리

병렬처리를 요하는 프로그램은 분산되는 Task가 고유의 데이타 영역 및 코드 영역을 소유하게 되고 이들은 각 프로세싱 노드의 메모리에 저장되어 수행되게 된다. 프로그램의 병렬처리를 위해서는 각각의 Task를 위한 임의의 프로세싱 노드가 할당되며 할당된 Task는 Physical Position이 정해지게 된다. 이들 Task들은 Task Force Manager에 의해 관리 되어지며 수행중에 필요한 Task들 사이의 통신은 시스템 소프트웨어에 의해 제공되는 기능을 이용하게 된다. 하나의 프로그램을 여러 프로세싱 노드에 분산시켜 병렬처리를 할 때 발생되는 문제점들에 대하여 알아보기로 한다.

제 1 절 분산 병렬처리의 기본 개념

1. 분산 병렬처리 단위

병렬처리를 위한 가장 기본적인 처리 단위를 Task라 할 때 Application 프로그램은 여러개의 Task로 구성되며 각 Task는 서로다른 프로세싱 노드에서 병렬수행될 수 있다. 이들 각 Task는 고유의 수행 환경을 갖게되며 처리되는 데이타 영역 및 수행 코드는 다른 Task와 독립되어 존재하게 된다. 또한 이들 Task는 필요에 따라 공유 데이타를 access할 수 있으며 이러한 데이타 공유는 시스템 소프트웨어에서 제공되는 기능을 이용하게 된다.

2. 병렬처리 프로그램

병렬처리 프로그램의 개발시에 우선적으로 고려되는 사항은 병렬처리의 기본 단위인 Task를 어떻게 나눌 것이며 이들 사이의 관계와 서로간의 통신 방법등에 대한 것이다. Task의 구분이 완료되면 이들 Task들을 논리적으로 연결하여 하나의 개념적인 Network가 생성되며 이 Network에 의해 각 Task 사이의 통신 형태 및 데이타 전송등의 Mechanism이 설정되게 된다. 프로그램 작성이 완료된 후에 병렬처리되는 각 Task들의 관계를 링크로 연결하면 프로그램의 관계가 그래프 형태로 표현되며 이 그래프는 각 Task 사이의 통신 형태를 보여주게 된다. 통신형태에서 나타나는 것은 두 Task 사이의 통신 횟수 및 개략적인 통신양으로, 이 데이타에 의해 분산된 각 Task들 사이의 통신을 하기 위한 최소한의 통신 성능을 예측할 수 있어야 하고 프로그래머는 시스템에서 제공되는 실제적인 통신 성능에 따라 프로그램을 수정하게 된다.

3. 타스크 간의 통신망

구현이 완료된 병렬처리 프로그램은 여러개의 Task로 구성되며 이들 Task들 사이는 상호 관계에 따라 데이타 통신이 이루어 지게 된다. 이들 각 Task사이의 통신이 존재하는 경우 이를 Link로 연결하면 한 프로그램에 대한 Task들 사이의 데이타통신 네트워크(Data Communication Network)를 얻을 수 있다. 이 관계도의 각 Link는 두 Task 사이의 데이타 통신이 존재함을 의미하며 또한 얼마나 많은 데이타 전달이 필요한가를 수치를 부여함으로써 표현할 수도 있는데 이 수치는 프로그래머에 의해 어느정도 예측이 가능하다.

여러개의 Task로 구성된 프로그램이 병렬처리 시스템에서 수행이 될때는 이들 각 Task가 하나의 프로세싱 노드를 할당받게 된다. Task에 대한 프로세싱 노드의 할당이 가장 이상적인 경우는 데이타 통신 네트워크의 각 Link와 프로세싱 노드 사이의 Physical Link 가 1대1로 Mapping되는 것이다. 그러나 병렬처리 시스템의 각 프로세싱 노드가 같는 통신 Link의 수는 한정되어 있으며(Transputer를 이용한 경우는 보통 4개) 따라서 한 노드와 연결 가능한 다른 프로세싱노드의 수도 제한되게 된다. 위에서 언급된 데이타 통신 네트워크는 각 Task들의 입장에서 볼때 적어도 한개 이상의 링크를 갖고 있으며 대부분의 경우는 프로세싱 노드가 갖고있는 Physical Link수보다 많다고 볼 수 있다. 즉 데이타 통신 네트워크를 그대로 Physical Link로 mapping이 불가능하게 된다.

4. 프로세싱 노드 할당

구현된 한 프로그램이 내포하게 되는 병렬처리를 위한 모듈들 사이의 관계 그래프는 실제 병렬처리 시스템의 각 프로세싱 노드에 각 Task를 할당하여야 한다. 프로세싱 노드의 할당은 프로그램 구현시에 특정 프로세싱 노드를 할당할 수 있으나 이러한 경우는 각 프로세싱 노드 사이의 통신 방법등에 대한 코딩 및 실제적인 통신에 대한 제어를 해야한다. 프로그래머가 직접 한 Task의 수행 노드를 지정하게 될 때 발생되는 가장 큰 문제점중의 하나는 개발 시스템의 하드웨어 구성과 같은 시스템에서만 수행 가능하다는 점이다. 이러한 문제점을 해결하기 위해서 Helios와 같은 병렬처리 시스템의 운영체제에서는 통신을 위한 기본적인 Mechanism을 제공하고 있으며 프로그래머는 Physical Communication에 대한 고려

를 하지 않고 단지 개념적으로 통신이 가능하다고 생각하고 프로그래밍을 할 수 있다.

위와 같은 역할이 가능한 병렬처리 시스템의 운영체제 중 Helios의 경우는 수행되고자 하는 프로그램으로부터 병렬처리를 위한 기본단위인 Task를 입력받아 사용가능한 프로세싱 노드에 임의의 Task를 할당하게 된다. 한 Task에 프로세싱 노드를 할당하는 방법은 Load-Balancer에 의해, 사용 가능한 프로세싱 노드들 중에 수행중인 Task가 없는 임의의 프로세싱 노드를 운영체제가 관리하는 테이블에서 찾아 할당하게 된다. 즉 프로세싱 노드를 할당하는 운영체제의 모듈은 임의의 한 Task로 부터의 프로세싱 노드 요청에 대해 전체 시스템의 성능을 최대화 하기 위해 각 프로세싱 노드의 로드를 고루 분산시키고자 하는 관점에서 프로세싱 노드의 할당이 이루어 진다.

이러한 운영체제의 기능에 따라서 프로그램의 각 Task 사이의 관계 그래프에 의해 표현되는 논리적인 통신 형태가 실제로 프로세싱 노드를 할당할 때 적용되지 않고있다. 즉 가장 이상적인 분산, 병렬처리 수행 형태에서의 병렬처리 가능성은 매우 희박하다고 할 수 있다. 여기서 가장 이상적인 프로세싱 노드 할당이란 프로세싱 노드가 갖고있는 통신 링크의 수에 따라 통신 관계가 있는 Task들 중에 통신양이 많은 Task들 사이를 우선적으로 Direct Link를 연결하여 줌으로써 전체적인 통신 cost가 가장 적은 경우를 의미한다.

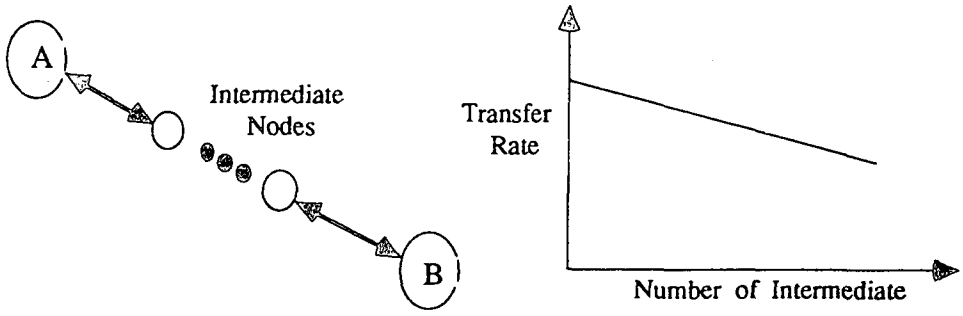
제 2 절 데이타 통신 특성

병렬처리 프로그램의 실행시에 각 프로세싱 노드 사이의 Interprocessor Communication은 프로그램의 Logical Communication 그래프에서 나타나는 통신 형태 이상의 복잡성을 갖게 된다. Logical Communication Graph에서는 한 Task에서 다른 Task와의 통신 Link 수의 제한이 없이 모든 관계가 표현될 수 있다. 그러나 실제 병렬처리 시스템에서의 수행 시는 프로세싱 노드의 링크 수가 한정된 관계로 모든 Logical Link에 대하여 Physical 한 Direct Link의 제공이 불가능 하다. 따라서 한 Task와 통신이 필요한 다른 Task와의 통신은 Direct Link에 의해서 보다는 중간 프로세싱 노드를 거쳐 통신이 이루어지게 된다. 이러한 경우 프로세싱 노드 A로 부터 프로세싱 노드 B로의 통신 사이에는 N(>= 0)개의 Intermediate Node가 존재하게 되며 Direct Link로 연결된 경우보다 통신 Cost가 높게 된다. 또한 Direct Link가 없는 경우는 통신 데이타의 Routing을 위하여 Intermediate Node의 프로세싱 power를 필요로 하게 되고, Routing 되는 데이타들 사이의 충돌에 의해 일부 Routing Path에 Bottleneck 현상이 발생될 수 있으며, 따라서 전체적인 프로그램의 성능에 많은 지장을 주게 된다. 위와 같이 Intermediate Node를 통한 프로세싱 노드사이의 통신에 있어서 Direct Link가 있는 경우와 비교하여 볼 때 다음과 같은 분야에서 프로그램 수행상의 통신 Overhead가 발생하게 된다.

1. 데이타 전송 지연 시간

중간 노드의 개입으로 인하여 단위 데이타 패킷의 전송 시간이 증가하게 된다. 중간 노드에서는 패킷의 전송을 위하여 메모리를 할당한 후 전송되는 데이타를 버퍼에 받고 전송이 완료된 후에는 다음에 전달해야 할 노드에 패킷 전송을

위한 초기화 작업 및 전송을 담당해야 한다. 이러한 중간 프로세싱에 의해 데이타 패킷의 전송은 전송시의 준비 절차가 완료되는 동안 지연되게 되며 중간 노드가 많을 수록 전송 지연은 커지게 된다. 아래의 그림-5.1은 중간 노드의 갯수에 따른 전체적인 통신 지연에 대한 실제 실험 데이타이다. (a)의 그림과 같이 두개의 프로세싱 노드 A, B 사이에 고정된 크기의 데이타 패킷 n개를 전송하게 하는 프로그램을 중간 프로세싱 노드의 갯수를 증가시켜가며 통신이 완료되는 시간을 측정하였다. 그결과 (b)에서와 같이 중간 노드의 수가 증가함에 따라 두 노드 A와 B 사이의 데이타 전송 시간은 점차 증가되었으며, 증가 폭은 거의 일정하였다. 이 실험 데이타의 결과를 볼 때 중간 노드의 수가 적을수록 두 프로세싱 노드 사이의 데이타 전송 시간의 지연이 최소화 될 수 있음을 알 수 있다.



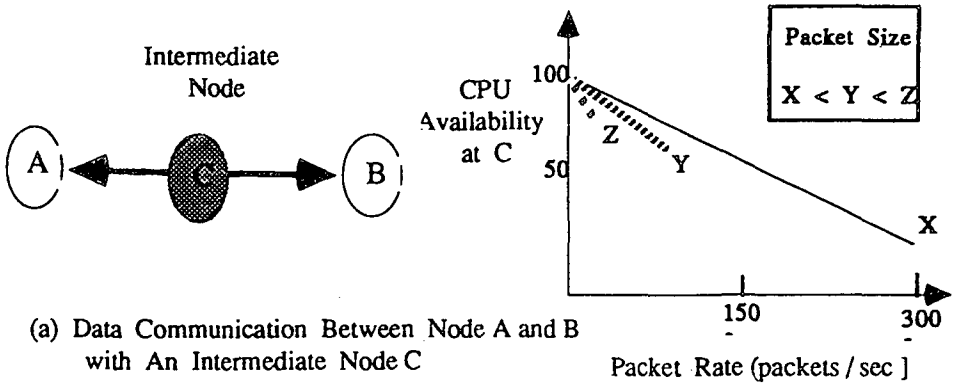
(a) Data Communication Between Node A and B with some Intermidiate Nodes

(b) Testing Result of (a)

그림-5.1. 중간 노드의 수와 Transfer Rate의 관계

2. 중간 노드의 라우팅 Overhead

중간 노드에서의 데이타 패킷 전달을 위한 일련의 절차가 이루어 지는 동안 수행되고 있는 Task의 수행은 일시적으로 중지된다. 데이타의 전달을 위해서 중간 노드는 시스템 함수를 이용하여 전달되는 데이타가 일시적으로 저장되는 메모리 버퍼를 할당하게 되고 또한 전달이 완료된 후에는 할당된 버퍼를 시스템 소프트웨어의 메모리 관리 모듈에 의해 돌려주어야 한다. 또한 노드 A로 부터 데이타 전송이 완료되면 버퍼에 저장된 데이타를 노드 B를 향하여 전달해 주어야 한다. 이러한 일련의 작업이 중간노드에게 어느정도의 Overhead가 되는가를 알아보기 위하여 실험을 해본 결과 그림-5.2와 같은 결과를 얻었다. 통신 Overhead를 측정하기 위하여 (a)의 경우와 같이 두개의 노드 A와 B 사이에 초당 KByte의 데이타 전송을 계속 반복하도록 하였으며 C 노드에는 일련의 Statement에 대한 Loop을 돌며 Loop 횟수를 counting 하였다. 그 결과 (b)와 같은 그래프를 얻을 수 있었다.



(a) Data Communication Between Node A and B with An Intermediate Node C

(b) Availability of the CPU for the process work at C

그림-5.2. Intermediate Node의 Routing Overhead 실험 결과

3. 링크 Bandwidth의 한계치 도달

실험에 사용된 Transputer 시스템의 Link Bandwidth는 14.5 Mbit/s 이다. 각각의 프로세싱 노드가 통신을 위해 사용하는 Link Path에 포함되는 임의의 Link중 하나라도 Link Bandwidth 한계에 도달하는 통신이 요구되면 이 Link로 인하여 전체적인 Performance에 막대한 영향을 미치게 된다. 즉 도심에서의 교통 소통과 관련하여 생각해 보면, 임의의 한 도로에 교통양이 많아 traffic Jam이 발생되게 되면 이 영향은 중위의 다른 도로에도 영향을 미치게 되어 전체적인 도심의 교통 소통에 지장을 초래하는 것과 마찬가지가 된다. 만일 하나의 Link에 Bottleneck 현상이 항상 존재하는 경우는 두 노드를 연결하는 새로운 Link를 연결해 줌으로써 이를 방지할 수도 있다. 이를 위해서는 특정 목적의 System Software의 보조를 받아야 하는데, 이때의 System Software의 역할은 두 노드 A와 B 사이의 Link에 대한 두개 이상의 Link로 통신 Request를 분산할 수 있어야 한다.

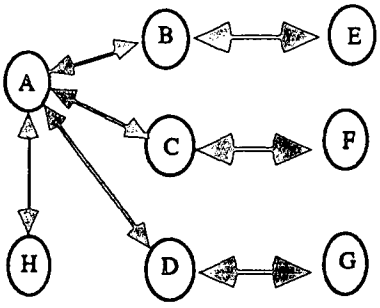
제 3 절 프로그램의 성능분석

1. 프로세서 할당과 프로그램 성능

앞에서와 같은 Interprocessor Communication 상에서의 Communication Cost 증가 요인들로 인하여 각 Task들 사이의 Intermediate Node 수는 병렬처리 프로그램에 여분의 Communication Cost의 증가를 유발하며 따라서 프로그램의 Performance 저하 요인이 되고 있다[Oehlrich]. 본 절에서는 여러개의 Task로 구성된 프로그램의 수행시에 이루어지는 Task에 대한 프로세싱 노드의 할당에 따라 프로그램의 Performance에 어떠한 차이가 있는가를 알아보기로 한다. Task에 대한 프로세싱 노드의 할당 결과 Intermediate Processing Node의 수가 많은 경우와 최소한으로 줄

인 경우에 있어서의 프로그램 수행 완료 시간을 점검해 보기로 한다. 본 실험에서는 최대 2개의 Communication Link를 갖는 8개의 프로세싱 노드를 각 Task에 임의로 할당한 Normal Case와 각 Task들 사이의 Intermediate Node 수가 최대한이 되도록 프로세싱 노드를 할당한 Worst Case, 그리고 각 Task들 사이의 통신을 위한 Path 상에 중간노드의 수를 최소한으로 줄인 프로세싱 노드의 할당 상태인 Best Case의 세가지 경우에 대하여 실험을 하였다.

Sample Program은 8개의 Task로 구성되어 있으며 이들 각각의 Task는 주어진 양 만큼의 데이타를 다른 Task에 보내는 작업을 하게 된다. 이 프로그램의 각 Task들 사이의 데이타 통신 관계 그래프 및 각 Task사이의 평균 통신 양은 그림-5.3.b와 같다. 표의 내용중 빈칸은 데이타 통신이 없는 것을 나타내며 주어진 수치는 Packet의 수를 나타내고 있다. 각 Packet은 동등한 크기이며 이 실험 데이타에서 사용된 Packet Size는 1M Byte 이다.



(a) Inter-Task Communication Style

	A	B	C	D	E	F	G	H
A	*	10	10	10				10
B	10	*			100			
C	10		*			900		
D	10			*			800	
E		100			*			
F			900			*		
G				800			*	
H	10							*

(b) Inter -Task Communication Pattern Tabel

그림-5.3. Inter-Task Communication of A Sample Program

그림-5.3.a와 같은 프로그램의 각 Task 사이의 Communication은 B와 E, C와 F, 그리고 D와 G 사이의 데이타 통신이 다른 Task와의 Communication 보다 상대적으로 많은 프로그램으로, 누가보더라도 3쌍의 Task들에 대해서는 서로 두 Task사이의 Communication을 위해 직접 연결된 Link가 존재하는 것이 Performance를 위해 좋다는 것을 쉽게 알 수 있다. 그림-5.4는 위에서 언급된 3가지의 Processing Node Allocation 형태에 대한 Physical Network Configuration을 보여주고 있다.

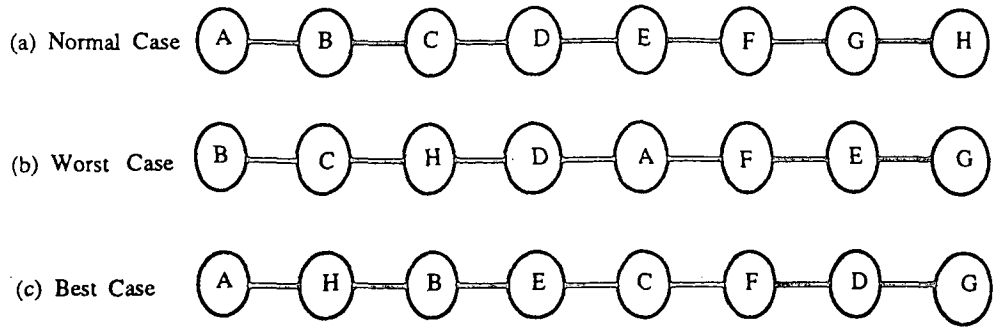


그림-5.4. Three Cases of Processing Node Allocation to Each Tasks

위의 세가지 Processing Node Allocation의 경우 각각의 프로그램 수행 완료에 소요되는 시간은 Worst Case인 경우와 Best Case인 경우에는 약 1.8배 가량의 시간이 더 소요되었으며, Normal Case인 경우는 Best Case에 비해 약 1.3배의 소요시간이 측정되었다. 이러한 실험 결과는 한 프로그램의 병렬처리를 위한 Processing Node Allocation의 중요성을 나타내고 있다. 이러한 결과는 현재 사용되고 있는 시스템 소프트웨어, 특히 Operating System에서의 Processing Node Allocation 방법에 문제로 볼 수 있다.

위와 같은 Intermediate Node에 의한 Inter-Processor Communication Overhad를 최소화 할수 있는 Processing Node Allocation이 가능한 운영체제가 제공된다면 프로그램의 Performance 저하를 최소화 할 수 있다. 이를 위해서는 모든 병렬처리 프로그램이 Task들의 Procesing Node Allocation에 대한 Task Distribution Network을 소유하고, Operating System에서는 이 Network을 이용한 Processing Node Allocation이 가능해야 한다.

2. 프로그램의 하드웨어 Dependency와 문제점

앞절에서 언급된 바와 같은 Operating System의 Support가 가능한 시스템에서 프로그래머는 Processing Node Allocation을 위해 필요한 Task Distribution Network을 생성해 주어야 하는데, 프로그램 설계시에 각 Task들 사이의 통신 형태를 예측할 수 있고, 또한 Sample Data에 의한 프로그램의 수행 결과를 분석함으로써 이 Network의 생성이 가능하다.

한 프로그램의 Task Distribution Network에 의한 Processing Node Allocation은 프로그램의 개발에 사용된 시스템에서 Inter-Processor Communication Cost를 최소화 할 수 있도록 되어있다. 그러나 개발이 완료된 프로그램이 실제 현장에서 사용될 때, 이미 생성된 Task Distribution Network이 적절한 가에는 몇가지 고려되어야 할 점들이 있다. 첫번째로는 개발 시스템과 실행 시스템의 하드웨어 구성 상의 차이가 있다. 프로세싱 노드들의 Link 수에 차이가 있는 경우에 작성된 Distribution Network은 사용할 수 없다. Transputer로 구성된 시스템에서도 Processing Node에 사용되는 Processor의 종류에 따라서 Link의 수가 4개(T800) 혹은 6개(T9000)가 될수 있기 때문이다. 또 다른 하나의 고려 대상은 프로그램이 실제로 다루는 데이타의 형태이다. 같은 프로그램이라도 현장에서 주로 사용되는 데이타의 형태에 따라 각 Task들 사이의 Communication Pattern은 차이가 있으므로 프로그램 개발시에 구성된 Task Distribution Network이 적절하지 않을 수 있다.

3. 데이타 통신과 시스템 성능과의 관계

앞에서 기술된 바와 같이 Interprocessor Communication에서의 문제점은 여러 개의 Task들로 구성된 병렬처리 소프트웨어의 Performance에 많은 영향을 미치게 된다. 한 프로그램이 여러개의 프로세싱 노드에 분산되어 병렬처리 되기 위해서 감수해야 하는 Communication Overhead는 각 Task들이 다른 Task와 통신을 하기 위해 거쳐야 하는 Intermediate Node 수에 따라 많은 차이가 있는데, 이 차이는 위에서의 실험 결과에 의해 쉽게 예측할 수 있다. 특히 멀티미디어 데이타 처리 프로그램의 경우는 다른 분야의 응용 프로그램에서 보다도 많은 양의 데이타를 처리해야 하며 따라서 각 Task 사이의 데이타 전송도 빈번하다. 따라서 Intermediate Node 수에 따른 전체적인 Communication Cost의 변화는 매우 크며, 이 Communication Cost는 최소화 하는 것이 바람직하다. 그러나 Communication Cost를 줄이기 위해 소요되는 Overhead가 감소된 Communication Cost보다 클 경우는 무의미하다.

현재까지 시스템의 Performance를 극대화 하기위한 Load Balancing에 대한 많은 알고리즘들이 연구되었고, 또한 Communication Cost를 최소화 하기위한 연구도 많이 수행되어 왔으나 실제 시스템에 적용하여 사용되기에는 자체 Overhead가 너무 큰 관계로 실효성이 없었다. 이들 대부분의 알고리즘들은 프로그램의 수행시에 항상 일련의 절차를 거쳐야 하므로 오히려 프로그램의 Performance에 방해가 되었다고 볼 수 있다. 본 보고서에서는 초기에 프로그램이 수행되는 일정 기간 동안만 프로그램의 수행 상태를 조사하여 해당 Site에 있어서 Communication Cost를 최소화 할 수 있는 Task Distribution Network을 재구성 함으로써 프로그램의 성능을 향상할 수 있는 모델을 제시한다.

여 백

제 6 장 분산, 병렬 검색 프로그램의 성능 분석

여 백

제 6 장 분산, 병렬 검색 프로그램의 성능 분석

현재까지 수많은 병렬처리 시스템들이 개발되었고 실제 현장에서 사용되고 있다.그러나 이들 시스템들은 주로 많은 양의 Computation Power를 필요로 하는특수목적용으로 사용되어 왔다. 대표적인 시스템 중의 하나는 Transputer를 이용한 시스템으로 수십개내지는 수백개 가량의 Transputer Chip을 이용한 Massively Parallel Architecture 시스템들이 많이 개발되었다. 최근에는 이들 병렬처리 시스템을 범용 시스템으로 사용하기 위한 많은 연구가 진행되고 있으며, 이들 연구의 가장 기본적인 것중의 하나는 많은 양의 Processing Node 및 Memory등의 Resource 관리등을 담당하는 운영체제라 할 수 있다.

현재 Transputer 시스템을 위한 운영체제에는 여러가지가 존재하나 완전히 독립된 형태로되어있는 것은 Helios 및 TDS 외에 많이 사용되고 있는 것이 없는 상태다. 그러나 이들 OS도 Transputer 시스템의 Hardware 기능을 충분이 활용 할 수 있도록 설계되지 못한 부분이 많고 아직 해결하지 못한 많은 분야가 존재한다.

특히 이들 시스템에서 하나의 프로그램을 여러개로 나누어 분산, 병렬처리를 하고자 할 때 과연 어느 정도의 response-time 증가를 제공할 수 있는가를 알아보고,또한 시스템의 전체적인 성능에 대한 bottleneck이 어디인가를 알아보기 위하여 한글 사전에서 하나의 단어를 찾는 4가지의 search 프로그램을 구현하여 search 속도를 분석하였다.

분석 결과는 하나의 프로그램에 의한 search 때보다 search 속도가 생각한 것보다도 훨씬 못 미치는 정도였는데, 이는 프로그램 구성상의 문제점 보다는 Transputer 시스템의 Architecutre 및 Operating System의 문제점이 더 크다고 할 수 있다.

Operating System의 문제점은 많은 사람들에 의해 제기되었으며, 또한 이에 대한 개선 방법도연구가 진행되어 일부에서는 Operating System의 일부 모듈을 변경하여 문제점들을해결해 가고 있는 실정이다. 또한 기존에 많이 사용되고 있는 Helios도 계속적인 Upgrade를 추진하고 있다.

본 장에서는 병렬처리 시스템의 가장 근본적인 추구 목적중의 하나인 분산, 병렬처리에 있어서의 시스템 bottleneck을 알아보기 위한 조사 과정의 하나로 선택된 4가지의 Search Program에 대한 알고리즘을 이용한 시스템 시험 방법과이 프로그램의 분산-병렬처리 방법에 대하여 각각 7.2장과 7.3장에서 기술하고수행 결과에 대한 분석 결과를 7.4장에서 기술하기로 한다.

제 1 절 분산-병렬처리 시험 방법

Multiprocessor Architecture System에서 하나의 프로그램을 여러 Processing Node에 분산하여 병렬처리를 하고자 할 때 어느 정도의 response-time이 향상될 수 있는가를 알아보고,현재의 운영체제(Multicluster-2용 OS로 사용되고 있는 Helios) 하에서 어떠한 문제점이 있으며,어떠한 부분이 시스템 성능 및 프로그램의

Response-Time에 대한 bottleneck인가를 알아보기 위하여 한글 사전으로 부터 단어를 찾는 Search 프로그램을 고려하기로 하였다.

프로그램의 전체적인 구성은 아래와 같다.

o 크기 *n byte*의 한글 단어 사전 (단순한 단어의 나열)

o 사전 구성을 위한 한글 문서 화일 다수

o Search를 위한 한글 문서 화일

o 각 Algorithm 별 Search Porgram

우선 크기 n-byte의 한글 단어 사전을 구성하기 위하여 임의의 한글문서 화일로부터단어를 읽어서 사전에 등록시키는 사전 구성기 프로그램을 coding하였다. 실제 한글사전은 각 단어에 대한 의미가 포함되어 있지만 여기서는 단순히 단어들만이 나열되어 있고 Search Program에서는 하나의 단어가 어디에 위치해 있는가를 File Pointer의 Offset 값으로 찾아낸다. 우선 사용하는 데이타의 양이 많아야 하므로 사전 구성기를이용하여 큰 크기의 텍스트 화일(사전)을 구성하였다.

위에서 구성된 사전으로 부터 Search를 위한 sample 텍스트 화일로부터 하나의 단어를 읽어서 구성된 사전에 존재 여부를 판단하고 있는 경우에 위치가 어디인가를 알아보는 Search 프로그램을 구현하였다. Search 프로그램은 4가지의 서로다른 Search Algorithm을 이용한 4개의 프로그램으로 구성되어 있으며 각각의 Algorithm 별로 시험이 이루어졌다. 사용된 4가지 Search Algorithm은 다음과 같다.

o Sequential Search

o Binary Search

o Fibonacci Search

o Interpolation Search

이들 Search 프로그램중 sequential search 프로그램은 standard I/O library(화일I/O를 위해 Buffering을 함) 함수 (fseek)을 사용 했고 그외의 프로그램들은buffering을 하지않고 'seek' system call을 이용하므로써 하나의 단어를 찾기 위해 하드디스크에 있는 사전 화일을 수십번 가량 직접 access 하도록 하여 데이타 communication 양을 늘렸다. 위의 4가지의 Search Program들에 대하여 각각 하나의 프로그램이 하나의 큰 화일에서 찾도록하여 병렬처리가 아닌 search와, 단어 사전을 n-개의 화일로 나누고, 각각의 화일에서 해당하는 단어가 있는가를 찾는 Search Program을 n-개의 Processing Node에서동시에 수행되도록 하였다. 후자의 경우는 n-개의 Processing Node중에 해당 단어를 찾은 Node의 실제 search 시간을 분산, 병렬처리 프로그램의 search 시간으로 하였다.

위와 같은 시험 환경을 구성하여 얻고자 한 것은 과연 Disk I/O가 많은, 즉 Disk를 하나의 Processing Element로 가정했을 때 병렬처리 프로그램에서 각 단위 프로세스 사이의 통신양이 많은 경우 분산-병렬처리의 효율이 현재 수행되고 있는 helios에서는 어느 정도가 되는가를 알아보는 것이다. 만일 원하는 효율이 나타나지 않을 때 어떠한 부분이 전체적인 성능의 bottleneck이 되는가를 알아보기 위한 것이었다.

분산 처리를 위해 10개의 분산 처리를 할 수 있도록 사전을 분산시켰고, 이들 각각의분산 사전은 각각의 search 프로그램들이 병렬처리를 하도록 하였다. 이들

프로그램들은 'Appendix-나'에 있다.

제 2 절 분산 병렬처리 방법

각 search progam은 다음과 같은 모듈들로 구성된다.

o master : 10개의 search program 관리

o worker : 지정된 하나의 사전에서 단어를 찾는 모듈

o Distributer : 위의 프로그램을 분산시키기 위한 CDL script 화일

위의 모듈중 'master'는 실제로 10개의 sub-dictionary에서 지정된 단어를 찾는 각 search program('worker')들과의 데이타 통신 및 제어 모듈로서, 초기에는 각 worker들이 이용할 사전을 알려주고, 시험 대상인 찾고자 하는 단어가 들어있는 화일을 읽어서 각 단어를 모든 worker에알려주고 단어를 찾은 worker로 부터 소요된 시간을 전송 받아 출력해준다. master는 각 worker의 동작을 제어하기 위하여 하나의 data structure를 이용한데이타 전송을 하게되고, 이 데이타를 받은 각 worker들은 지시된 단어를 찾게된다.

worker는 master에서 보내온 데이타로 부터 자신이 찾아야할 사전이 무엇인가를 알아내고 정해진 algorithm에 의하여 주어진 단어를 찾는다. 단어를 찾기전에 시간을 count하기 위한 초기작업을 하고 사전 search가 완료되면 그동안의 소요 시간을 계산하여 정해진 구조체에 save하여 master에게 보낸다. 위의 작업이 완료된 각 worker는 master로 부터 요구되어지는 새로운 단어를 받을 때까지 일시 중지되며, 새로운 단어를 받은 후에는 위의 작업 과정을 반복한다.

Distributer는 master와 worker로 구성되는 병렬처리 프로그램을 어떻게 분산처리할 것인가를 기술한 화일로서 Helios에서 제공하는 CDL(병렬처리를 위한 Component Distribution Language)로 작성되며 worker의 개수, master와 worker의 통신 channel 구성, 분산 형태등을 기술하고 있다. 이 화일은 4가지 Search Program마다 각각 하나씩 존재하며 각 Search Program은 이 Distributer모듈을 수행함으로써 분산 search가 시작된다.이 Distributer는 보통의 ASCII Text 화일로 되어있으며 Helios의 Shell mode중 'CDL mode'에서수행될 수 있는 것이다. 이 모듈을 'CDL mode'에서 수행할 수도 있는데, 이를 위해서는 CDL compiler에 의해 생성되는 실행 가능한 object 화일을 만들어야 한다.

현재 16개의 Transputer로 구성되어있는 Multicluster 2에서는 booting시에 network configuration이이루어 지는데 본 search 프로그램들의 search 속도가 Network configuration에 따라 다소 다를 수있으나 우선은 search 프로그램의 분산, 병렬처리 결과를 보기위하여 Network configuration은 고려하지 않았다. 10개의 worker들은 16개의 Transputer에 분산되어 수행되는데 이들 분산 과정은 Helios에서 제공하는 Load-Balancer에 맡겼다.

제 3 절 분산 병렬처리 결과 분석

결론부터 말하면, 분산-병렬 search 프로그램들의 결과는 예상보다는 효과가 크지않은 것으로 나타났다.병렬처리를 하지않고 n byte로 구성된 사전에서 하나의 단어를 찾는 프로그램의 search 속도나 n/10 byte의 분리된 사전을 10개의 프로그

램에 의해 병렬 search한 경우의 속도 차이가 별로 없었다.

각각의 search program에 대한 사전은 동일한 것을 사용했으며, 사전의 크기는 1459단어로 구성되었고 2-byte 완성형 코드로 sorting 되었다.

4가지의 search program중 sequential search를 제외한 3가지는 사전에 대한 access시에 seek() system call을 사용한 관계로 standard I/O library를 이용한 sequential search 프록그램에 비하여 search 시간이 많이 걸렸다. 각 search program의 한 단어에 대한 평균 search time을 보면 아래의 표와 같다.

Algorithm	Single Process Search	Parallel Search
Sequential Search	10.5	6.0
Binary Search	318.8	178.0
Fibonacci Search	435.6	209.0
Interpolation Search	470.1	526.0

위의 자료는 Parallel Search의 경우 10개의 process로 분산되어 병렬수행된 경우이고Single Process Search는 하나의 process에 의한 결과이다. 위의 테이블에서 나타난 결과를 보면 쉽게 알 수 있듯이 10개의 프로세스로 분산되어 수행되었을 때 적어도 7-8배 가량의 speed-up이 있어야 하나 실제적으로는 약 2배 정도의 속도만 향상된 것을 알 수 있다. Sequential Search의 경우는 standard I/O library를 이용하여 프로그램된 관계로 위에서와 같이 다른 3가지의 Search Program에 비하

여 매우 빠른 속도로 search가 수행되었으나 역시 Single Process에 의한 search와 10개의 process에 의한병렬 search의 속도 차는 약 2배에 지나지 않았다.

위와 같은 결과를 미루어 병렬처리를 위한 시스템의 하드웨어 혹은 System 소프트웨어상에어떠한 문제점이 있음을 쉽게 발견할 수 있다. 즉 여러개의 process에 의한 병렬처리 효과가 어떤 이유로 인하여 나타나지 않고 있는 것이다. 물론 위의 결과가 Transputer 시스템 자체에 하드디스크가 없어서 그렇다고 할 수 있으나 2가지의 경우 똑같은 환경이므로 고려 대상이 아님을 알 수 있다.

그러면 위의 테이블에서와 같은 결과가 나타난 이유는 무엇인가? 결국은 사전을 acess하는 행위(Disk I/O operation)가 시스템의 overhead를 증가시켰고 이 overhead로 인하여 시스템의 전체적인 성능저하를 가져오게된 것이다. 즉 사전을 access하기 위한 request 및 사전 데이타의 전송이 각 process에 전달되기 위해서는 중간에 있는 node를 지나야 하는데, 이를 위해서는 중간에 있는 Transputer 노드는 데이타를 전달시켜야 하고 이로 인하여 이 Transputer 노드는 자신의 search program 수행이 일시적으로 중지되어야 한다. 이러한 일시적인 node의 사용 중지가 전달해 주어야 하는 데이타의 양이 적은 경우는 시스템의 성능에 큰 영향을 미치지 않겠지만 위의 프로그램에서와 같이 사전에 대한 access가 매우 빈번한 경우는 자신이 수행해야 하는 search program을 위한 시간을 많이 빼앗기게 된다.

이러한 Data Communciation Overhead가 위의 테이블에서 보는 바와 같이 매우 심각한 상태임을쉽게 알 수 있다. 위의 같은 결과를 그림으로 내타내 보면 아래와 같다.

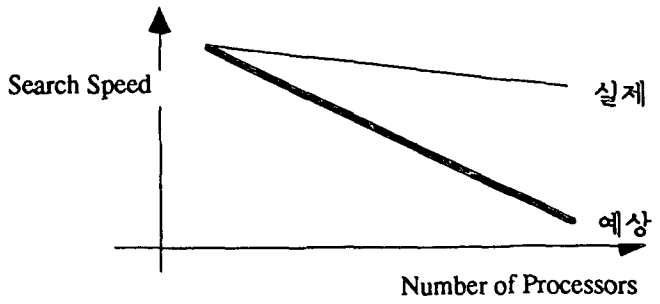


그림-6.1. 병렬처리 속도의 예상과 실제

즉 병렬처리 결과의 예상과 실제가 매우 큰 차이가 있음을 알 수 있다. 이러한 결과는 위와 같은 search program에서 뿐만 아니라 communication data 양이 매우 많은 경우에는 위의 표에서와 같은 결과가 나타날 것으로 생각되며, 특히 병렬처리 시스템이 General Purpose 시스템으로 이용되려는 추세가 시작되는 현 시점에서 위와같은 문제점은 가장 먼저 해결해야 할 것중의 하나라 할 수 있다. 특히 수십 혹은 수백개의 node를 갖는 병렬처리 시스템에서 수행될 경우에는 중간에 통과 해야 하는 node수가 더욱 많아질 것이며 따라서 위와 같은 결과 그 이상으로 시스템의 성능이 저하될 우려가 있다. 이를 위해서는 위와 같은 문제점이 해결되어야 하는데 이에 대한 해결 방향은 하나의 프로그램이 병렬처리될 때의 process 사이의 communication 양에 대한 graph로 부터 가장 communication overhead를 줄일 수 있는 형태로 processor node의 할당이 이루어지도록 시스템 소프트웨어가 동작되어야 한다.

여 백

제 7 장 적응 분산 맵 자동 생성 모델

여 백

제 7 장 적응 분산 맵 자동 생성 모델

본 장에서는 구현하고자 하는 멀티미디어와 같은 대용량의 데이타를 처리하는 프로그램의 분산, 병렬처리를 위한 타스크 분산 맵 자동 생성 소프트웨어의 개략적인 동작 방법에 대하여 기술한다.

이 모델은 어떠한 병렬처리 프로그램을 수행함에 있어 각 단위 수행모듈등에게 프로세싱 노드를 할당할때 노드들 사이의 통신 overhead를 최소화 할 수 있는 프로세싱 노드의 연결관계 맵을 자동으로 생성함으로써 프로그램의 성능을 극대화함을 목적으로 한다.

제 1 절 시스템 구성도

이제까지 연구되어진 내용을 바탕으로 Interprocessor Communication에서의 문제점은 여러개의 타스크들로 구성된 병렬처리 소프트웨어의 성능에 많은 영향을 미치게 된다. 한 프로그램이 여러개의 프로세싱 노드에 분산되어 병렬처리되기 위하여 감수해야하는 Communication Overhead는 각 타스크와 통신을 하기 위해 거쳐야하는 Intermediate Node의 수에 따라 많은 차이가 있다. 이러한 문제점을 개선하기 위하여 2개의 임의의 Processing Node간의 Intermediate Node의 수를 최대한 줄이는 것이 개발하고자 하는 시스템 소프트웨어의 목적이다.

그림-7.1은 기본적으로 시스템을 구성하는 각각의 모듈들을 개념적으로 병렬처

리 시스템의 일반적인 운영체제내에서의 실행환경을 바탕으로 도식화 한 것이다. 한 Application의 Program image는 그 Application을 구성하는 실행단위인 여러개의 Task들로 구성되어 있고 또한 이들 Task들을 병렬처리 시스템 상에서 어떠한 방법으로 분산처리 되어지는가에 대하여 기술되어 있는 Task Distribution Network 테이블를 포함한다. 본 개발하고자 하는 시스템 소프트웨어의 역할이 가장 최적의 기능을 갖은 Task Distribution Network를 구성하는 것이다.

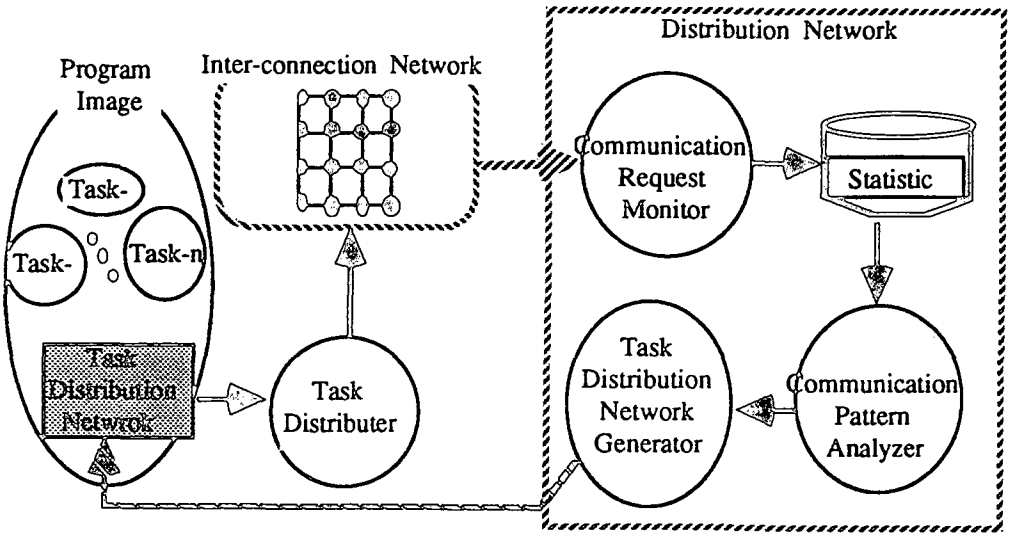


그림-7.1. The Conceptual Structure of Adaptive Task Distribution

이러한 구성요소를 갖은 Program Image를 바탕으로 기본적으로 병렬처리 시스템 운영체제에서 제공되어지는 모듈인 Task Distributer는 사용가능한 Processing Node Network에 Program Image를 Mapping 시킴으로써 실행하고자 하는 Application의 Inter-connection Network를 구성한다. 이러한 procedure를 거침으로써 비로서 Application의 실행이 가능해지는 것이다. 위에서 기술되어진 실행환경을 바탕으로,

개발하고자하는 시스템 소프트웨어를 Integratc하는 방법이 그림-7.1에서 볼 수 있듯이 사각형 박스안에 있는 내용들이다. Communication Request Monitor는 각각의 task들이 수행되면서 서로 다른 task들과 데이타를 주고 받을때 이러한 내용들을 수집하여 각각의 task들의 통신 Statistic Data File (logging file)을 생성한다. 이러한 logging file을 생성함으로써 발생하는 Overhead는 특정 Application이 실제현장에서 적응이 완료되었을시에는 모두 제거되어짐으로써 불필요한 사항들을 없앨 수 있다.

이 logging file을 바탕으로 Communication Pattern Analyzer는 각각의 task entry 별로 task간의 통신 상태를 나타내는 Communication Pattern Table을 만들어낸다. 이 Table은 각 Task 사이의 통신 양에 대한 Table로서 단위 시간당 요청된 Communication Request의 평균치이다. 이 Table 외에도 단위 시간당 Communication Request가 정해진 threshold Value를 넘은 Link도 분석해 낸다. 생성된 Communication Pattern Table은 그림-5.3.b와 같은 형태이며, Link Bandwidth의 한계치에 근접하는 Link들은 하나의 List로 구성된다. Link Bandwidth를 넘거나 근접하는 Communication Request가 요청되는 Link는 프로그램 Performance의 Bottleneck이 될 가능성이 많기 때문에 이 Link를 통하게 되는 두 Task 사이의 통신 Path를 변경시켜줌으로써 Link의 이용을 줄여야 하기때문이다.

이렇게 하여 생성된 Communication Pattern Table과 Link Bandwidth의 한계치에 근접하는 Link들의 List는 Task Distribution Network Regenerator에게 넘겨져 마지막으로 현재 사용중인 Task Distribution Nertwork Table보다 효율적인 것을 만드는데 사용된다.

하는 것을 목적으로 한다. 프로세싱 노드의 physical link를 통한 Task간의 통신방법에는 여러가지 종류가 있는데 이것을 본 소프트웨어 개발시스템인 Multicluster/2의 운영체제인 Perihelion사의 Helios 에서 사용되는 방법에 관하여 알아본다.

Helios환경하에서는 4가지 방법으로 Task들이 서로 통신할 수 있는데 이것을 종류별로 알아보면 다음과 같다.

- Direct : Direct link usage in-line assembler macros.
- Primitive : Message passing primitives (GetMsg() and PutMsg()).
- System : System library functions (Read() and Write()).
- posix : Posix library functions (read() and write()).

이러한 mechanism은 실제로 프로그래머가 모두 사용할 수 있지만 Direct와 Primirive를 이용할때에는 데이타 전송시 physical link에서 발생할 수 있는 error를 recovery하여준다던가 데이타 전송시 필요한 buffer 관리, hand shaking mechanism등을 모두 프로그래머가 코딩하여야만 하는 사용방법의 편이도 부족 때문에 이러한 기능들을 프로그래머 대신 하여주는 System, posix library function이 많이 쓰인다. 물론 이러한 function을 사용할때 생기는 overhead에 대하여는 감수 하여야만 한다. Posix Library function인 read()와 write()의 syntax는 다음과 같다.

```
Syntax : read  (link_number, buffer_address, size_of_buffer, timeout);
         write (link_number, buffer_address, size_of_buffer, timeout);
```

이와 같은 환경하에서 Communication Request Monitor를 구현하기 위하여 필요한 것은 read 및 write function을 사용할때 logging file에 저장하여 할 데이타를 어

제 2 절 병렬처리 시스템 요구사항

위와 같은 Model을 위한 Parallel Processing System은 몇가지의 요구사항을 만족해야 한다. Hardware 및 System Software의 요구 기능은 다음과 같다.

- o Distributed Memory Architecture
- o Dynamic Reconfiguration of Physical Links
- o Communication Statistic Gathering Function of Operating System

각 프로세싱 노드는 Local Memory를 소유하고 있어야 하며, Processing Node의 Physical Link의 연결에 의한 Processor Network Configuration이 프로그램의 수행전에 이루어져야 하므로 소프트웨어에 의해 프로그램의 수행전에 Physical Link에 대한 연결이 가능해야 한다. System Software는 Link 사이의 통신 형태에 대한 Monitoring이 가능해야 한다. 즉 physical Link를 통한 데이타 전송이 요청될 때마다 요청된 데이타의 크기 및 요청 시기, 요청 Processor 번호 및 Destination Processor 번호에 대한 Logging이 가능해야 한다. 또한 이 기능은 사용자의 요구에 따라 수행되지 않을 수 있어야 Adaptation이 완료된 프로그램의 수행시에 불필요한 Overhead를 제거할 수 있다.

제 3 절 통신 요구 감시기

이 모듈은 Task간의 통신요구가 있을때마다 이러한 통신요구 내용들을 logging file에 저장함으로써 다음과정(Communication Pattern Analyzer)에 필요한 정보를 제공

떠한 방법으로 수집하는 것에 관한것이다. 이것은 Helios의 소스를 분석하여 read 및 write function을 필요한 부분을 변경하여 구현하는 방법도 있겠지만 이것보다는 각각의 function에 해당하는 pseudo function을 제작하여 program 작성시 사용하는 방법이 조금더 용이 할 것이다. 이들의 peusdo function의 syntax를 알아보자.

Syntax : lread (link_number, buffer_address, size_of_buffer, timeout);

lwrite(link_number, buffer_address, size_of_buffer, timeout);

이들 function들은 Posix library 에서 제공하는 read /write와 같은 파라미터를 갖는다. 그럼 이들 function들의 역할에 관하여 살펴보자.

```
/* lread styled C */
int lread (link_nb, buf_add,  size_buf, timeout)
int link_nb;
char *buf_add;
int size_buf;
int timeout;
{
    if (LOG_FLAG)
        Communication_Request_Monitor(link_nb, size_buf);
    read (link_nb, buf_add, size_buf, timeout);
}
```

```
/* lwrite styled C */
int  lwrite (link_nb, buf_add,  size_buf, timeout)
int link_nb;
char *buf_add;
int size_buf;
int time9ut;
{

      if (LOG_FLAG)
            Communication_Request_Monitor(link_nb, size_buf);
      write (link_nb, buf_add, size_buf, timeout);
}
```

여기서 사용되는 LOG_FLAG라는 프래그는 실행하고자 하는 application이 실제로 현장에서 적응된 후에는 제거되어저야만 불필요한 overhead를 없앨 수 있다.

이렇게 하여 얻어진 information을 logging file에 저장함에 있어서 필요한 data structure가 있는데 이 data structure를 다음과 같이 나타내었다.

```
logging data :  struct log_data
{
     int  sourceID;
     int  destID;
     time act_time;
     int size;
}
```

Communiation Request Monitor에서는 파라미터로 전달되어진 사용된 link의 번호와 데이타의 크기를 수집하고 현재의 시스템 시간과 source processing node를 log_data라는 data structure를 이용하여 logging file에 써넣는다. 다음은 logging file의 format이다.

logging file format ==> current time : source ID : destination ID : size of buffer

이들 데이타는 정해진 기간동안에 계속 축적되며, 일정 기간이 경과한 후에 'Communication Pattern Analyzer"에 의해 Communication Pattern Table을 생성한다.

Communication Request Monitor의 일반적인 절차를 다음과 같이 나타낸다.

Communication_Request_Monitor (,,,)

Input : link number, size of buffer
Output : logging file

[Step-1] fetch the Destination node via link number

[Step-2] fetch the size of data to transfer via size of buffer

[Step-3] fetch the Source node by Distribution Network Table

[Step-4] fetch the System Time

[Step-5] If logging file not exist
Create logging file
else open logging file

[Step-6] write these information in logging file

제 4 절 통신 패턴 분석기

Communication Request Monitor Module에 의해 수집되는 이들 데이타(logging file)는 정해진 기간동안에 계속 축적되며, 일정 기간이 경과한 후에 "Communication Pattern Analyzer"에 의해 Communication Pattern Table을 생성한다. 이 Table은 각 Task 사이의 통신 양에 대한 Table로서 단위 시간당 요청된 Communication Request의 평균치 이다. 이 Table 외에도 단위 시간당 Communication Request가 정해진 threshold Value를 넘은 Link도 분석해 낸다. 생성된 Communication Pattern Table은 그림-5.3의 (b)와 같은 형태이며, Link Bendwidth의 한계치에 근접하는 Link들은 하나의 List로 구성된다. Link Bendwidth를 넘거나 근접한는 Communication . Request가 요청되는 Link는 프로그램 Performance의 Bottleneck이 될 가능성이 많기 때문에 이 Link를 통하게 되는 두 Task 사이의 통신 Path를 변경시켜줌으로써 Link의 이용을 줄여야 한다. Communication Pattern Analyzer의 일반적인 절차를 다음과 같이 나타낸다.

Communiation Pattern Analyzer (,,,)

Input : logging file
Output : Communication Pattern Table,
list of links (exceeding link bottleneck)

[Step-1] Open logging file

[Step-2] Initialize Communication Pattern Table
(Communiation Pattern Table [nb_of_tasks, nb_of_tasks]),
and list of links

[Step-3] Read un element in logging file

[Step-4] If EOF goto Step-7

[Step-5] Update Communiation Pattern Table

[Step-6] Goto Step-3

[Step-7] For all pair of tasks via Communication Pattern Table
Check if exists a link exceeding link bandwidth
If EXIST then insert it in the list
Next

[Step-8] Close logging file

[Step-9] End

제 5 절 타스크 분산 맵 생성

Communication Pattern Analyzer에 의해 생성된 Communication Pattern Table과 각 프로세싱 노드가 갖고있는 Link수 등을 파라미터로 하여 Task Distribution

Network Regenerator는 Communication Cost를 최소화 할 수 있는 새로운 Task Distribution Network을 생성하게 된다. 이 모듈은 크게 4개의 procedure로 구분되어 진다. 이것을 다음과 같이 표현한다.

```
/*  Task Distribution Network Generator styled C */

dist_network_type task_distribution network_generator ( pattern_table, number_of_link)
pattern_table_type pattern_table;
int number_of_link;
{

        for ( all pair of tasks  via pattern table)
        {
                allocation_of_processing_node();
                linking_two_processing_node();
        }

        for (all link used)
        {
                checking_link_bandwidth();
                if(bottleneck checked)
                        removing_bottleneck();
        }
```

return(new_task_distribution_network);

}

위의 내용을 조금 더 구체적으로 알고리즘 형태로 다음과 같이 나타낸다.

Algorithm-1 :

Task_Distribution_Network_Generator (, , ,)

Input : Communication Pattern Table
Number of connection links of each processing

[Step-1] Sorting the Communication pattern Table entries

[Step-2] Select a table entry which has the largest values

If there is no tabel entry which is not concerned
then Go To [Step-6]

[Step-3] Processing Node Allocation (Show Section 1.)

[Step-4] Linking Two Processing Nodes (Show Section 2.)

[Step-5] Go To [Step-2]

[Step-6] Checking whether there is any link
which is possible to be a communication bottle-neck
(Show Section 3)

[Step-7] Reduce the Total Communication Rate of the link
which is possible to be a communication bottle-neck
(Show Section 4.)

1. 프로세싱 노드 할당

새로이 고려되는 Table Entry에서 두 프로세싱 노드중 존재하지 않는 프로세싱 노드에 대해서는 새로운 프로세싱 노드를 할당한다. 만일 두 노드 모두 존재하면 두 노드 사이의 통신 경로를 제공하기 위한 Linking 과정으로 진행된다.

2. 두개의 프로세싱 노드 연결

두 프로세싱 노드 사이의 Communication을 위한 Link의 연결은 두 프로세싱 노드가 사용하지 않는 link를 보유하고 있는가에 따라 구분 된다. 즉 2 프로세싱 노드 모두 사용하지 않는 Link가 존재하는 경우와 두 프로세싱 노드중 한 노드만 존재하는 경우, 그리고 두 프로세싱 노드 모두 사용하지 않는 Link가 없는 경우로 구분할 수 있다. 첫번째의 경우는 두 노드를 임의의 사용하지 않는 두 Link를 이용하여 연결하고, 두번째 및 세번째의 경우는 사용하지 않는 Link가 없는 노드와 Directly Connected되어 있는 주위의 노드에 대하여 위와 같은 경우를 다시 고려하면 된다. 즉 이 Algorithm은 Recursive Call에 의해 쉽게 Implement될 수 있다. 두 프로세싱 노드의 Linking Algorithm에 대한 자세한 내용은 다음과 같다.

Algorithm-2 :

Linking_Two_PE (From_PE, To_PE)

[Case-1] When each of From_PE and To_PE has any unused link

⟹ *Linking From_PE and To_PE using their unused links Link*

[Case-2] When only one of two processing nodes has any unused link
(Let No_Link_PE be the processing node which has no unused link)

[Case-2-1] If Existed_PE has unused link

⟹ *Get a set of processing nodes which are directly linked with No_Link_PE*

If there is a node in the above set which has any unused link,
then linking two nodes with the unused link. & end-of-procedure

For all processing nodes
Select a processing node (New_PE) from the above set;
Call Linking_Two_PE(New_PE, To_PE);
Until above recursive call is succeeded.

[Case-3] When both two processing nodes have no unused link

Get two set of processing nodes which are directly connected
with From_PE and To_PE
(Let's call them as From_PEs and To_PEs respectively)

Select two processing nodes from From_PEs and To_PEs
(Lets call them New_From_PE and New_To_PE respectively)

If there is any pair of processing nodes (New_From_PE, New_To_PE)
which two processing nodes have their own unused link
then linking two nodes & end-of-procedure

For all pairs of (New_From_PE, New_To_PE)
Call Linking_Two_PE(,)
Until above recursive call is succeeded.

3. 통신 링크의 병목상태 추출

Task Distribution Network 생성 Algorithm의 [Step-5] 까지의 수행결과 하나의 Inter-Processor Communication Network을 얻을 수 있다. 이 Network의 생성은 임의의 Table Entry 하나의 관점에서 두 Task 사이에 Intermediate Node의 수를 최소화하기 위한 결과로서 전체적인 측면에서의 Communication Complexity는 고려되지 않았다. 즉 위와 같이 구해진 Task Distribution Network에서는 임의의 한 Link가 여러 Task 사이의 Communication Path에 포함됨으로 인하여 Link Bandwidth 이상의 Communication Request가 있을 수 있다. 이러한 점을 개선하기 위해서는 Link Bandwidth에 가까운 Communicaiton Request가 있을 가능성을 점검하고, Communication Bottleneck의 가능성이 있는 Link에 대해서는 이 Link를 사용하는 임의의 두 Task 사이의 통신 Path를 변경해 주어야 한다. 이를 위해서는 우선 모든 Link에 대한 Communication 양을 구해야 한다. 임의의 두 Processing Node I와 J를 연결하는 Link (I, J)에 대한 Communication 양은 Task(X)와 Task(Y)의 Communication Path가 Link(i,j)를 포함하는 모든 X, Y에 대하여 Communication Pattern Table의 모든 (X,Y) 및 (Y,X)의 Communication 양을 더한 것이다. 따라서 임의의 processing node I와 J를 연결하는 Link(i,j)에 대한 Total Communication 양 (TC_{ij})는 다음과 같이 정의 된다.

$$TC_{ij} = \Sigma_{X,Y} \left(C(X, Y) + C(Y, X) \right)$$

where X, Y is for all X, Y, where the communication path between

Task-X

and Task-Y includes the Link(i,j)

C(X,Y) is the Communication Rate Between Task-X and Task-Y

위의 식에 의해 구해진 각 Link의 Total Communication 양이 일정한 크기 이상(Threshold Value < Link Bandwidth)을 넘는 경우에는 이 Link에 Communication Bottleneck 현상이 발생할 가능성이 있다. 그러므로 이러한 모든 Link에 대해서는 TC_{ij}의 값을 감소시켜야 한다.

4. 통신 링크 병목 현상 제거

Total Communication Rate가 Threshold 값을 넘는 모든 Link는 Communication Bottleneck이 될 가능성이 있으므로 TC_{ij} 값을 감소시켜야 하는데, 이를 위해서는 다음과 같은 과정을 거쳐야 한다. 아래의 Algoritm에 의해 수정된 Task Distribution Network은 Communication Bottlenecck의 발생 가능성을 최소화한 최종 Output이다. 이렇게 구해진 Network은 기존에 존재하는 Network을 대치함으로써 해당 Site에서의 Adaptation이 가능하게 된다.

Algorithm-3 :

Removing_Communication_Bottleneck(Link(I, J))

[Step-1] Get a set of C(X, Y), where the communication path between Task-X and Task-Y includes the Link(I, J)

[Step-2] Sorting the set with the value of C(X, Y)

[Step-3] Scanning the set from the largest one satisfying the following condition:
<Cond> Summation of all scanned C(X, Y) < Threshhold value
(Let C(M, N) be the element of all C(X, Y) not scanned yet)

[Step-4] For all C(M, N),
Call Linking_Two_PE (M, N) but Link(I,J) must be avoided.

5. 통신 링크 병목 현상 감시

앞에서 제시된 Algorithm들에 의해 병렬처리 프로그램의 Task Distribution Network을 생성한 후에는 이에 대한 Tunning 과정을 거치게 된다. 즉 도출된 Task Distribution Network에 의해 프로그램을 분산 병렬처리 하면서 각 Processing Node를 연결하는 Physical Link의 Bottleneck만을 조사하게 된다. 즉 구해진 Task Distribution Network은 Communication Bottleneck의 발생 가능성을 최소화한 것이므로 실제 프로그램의 수행시에 Communcation Bottleneck의 발생 가능성은 역시 남아있다. 이때 부터는 Task 사이의 Communication Request를 감시하는 것이 아니고 단지 Physical Link에 대한 Communication Request가 Link Bandwidth를 초과하는가 만을 감시한다. 만일 임의의 Link에 대한 Communication Bottleneck이 발생하

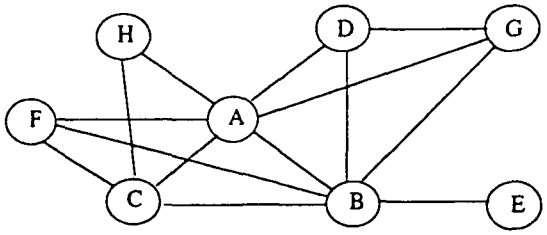
였을 경우는 Algorithm, Removing_Comm_Bottleneck()을 이용하여 Task Distribution Network을 재구성 한다.

위와 같은 감시가 어느정도 지속되는 동안에 Communication Bottleneck이 발생되지 않으면 그림-7.1중에서 "Adaptive Task Distribution Network Regeneration" Module들의 수행을 중지함으로써 이로 인한 프로그램의 Performance 저하를 제거하게 된다. 위와 같은 모든 과정이 완료되면 한 프로그램의 Site Adaptation 과정은 끝나게 된다.

제 6 절 시험 결과

그림-7.2는 제안된 알고리즘을 적용한 예제이다. 그림-7.2.a는 sample program의 논리적 Inter-Task Communication Network을 나타내고 그림-7.2.b는 이것에 의하여 Communication Pattern Analyzer가 생성한 Communication Pattern Table을 나타낸다. 그림-7.2.c는 communication bottleneck을 찾아내기전 도중의 task distribution network을 나타낸다. 그림-7.2.d는 communication bottleneck을 제거한 마지막으로 구해진 task distribution network을 나타낸다. 이렇게 하여 구하여진 network의 성능을 조사하기 위하여 두가지 다른 방법으로 sample program의 response time을 측정하였는데, 하나는 본 연구에서 제안된 알고즘을 적용하였고 다른 하나는 사용된 운영체제(Helios Operating System에서 제공된 사항만을 적용하였다. Helios 운영체제에서 제공되어지는 사항만을 적용할때는 한 프로그램을 실행할때 필요한 타스크들을 각각의 실행 프로세서로 할당시, Inter-Task Communication Network와 관계없이 physical network상의 프로세서를 할당하기 때문에 여러 link에서 Communication bottleneck이 발생하여 프로그램의 성능이 상당 부문 저하되는 것을 볼 수 있었다. 그러나 본 연구에

서 제안된 모델을 적용하였을 경우에는 평균적으로 30%의 프로그램의 성능 향상을 도모할 수 있었다.

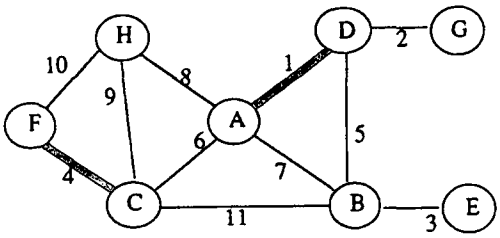


(a) Inter-task communication network of sample program

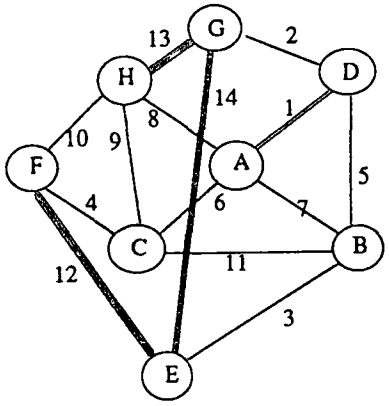
	A	B	C	D	E	F	G	H
A	*	500	600	950	0	300	140	430
B	800	*	210	670	850	180	120	0
C	600	230	*	0	0	800	0	430
D	900	700	0	*	0	0	880	0
E	0	890	0	0	*	0	0	0
F	350	200	830	0	0	*	0	0
G	120	210	0	900	0	0	*	0
H	630	0	450	0	0	0	0	*

(b) Communication Pattern Table of (a)

그림-7.2(a,b). An Example of applying proposed algorithms



(c) Task Distribution Network
with communication bottlenecks



(d) Final Task Distribution Network

그림-7.2(c,d). An Example of applying proposed algorithms

제 8 장 결 론

여 백

제 8 장 결 론

어떠한 타입의 병렬처리 시스템이라도 각각의 노드들을 연결하여 주는 물리적인 연결방법은 한정되어 있기 때문에 각각의 프로세싱 노드들을 전체적인 시스템 측면에서 모두 연결하여 주는 것은 불가능한 일이다. 이러한 이유로 어떠한 프러세싱 노드와 다른 프로세싱 노드가 서로 통신하고자 할때는 필요에 따라서 몇개의 중간노드를 거쳐야 하는 경우가 생기는데 이러한 경우에는 각 노드들이 실행하는것 보다 통신에 더 많은 시간을 사용함으로써 상대적으로 프로그램의 성능을 저하하는 요소로 작용한다. 따라서 병렬처리 시스템에서의 프로그램 성능은 그것을 구성하는 각각의 기본 실행요소들을(task) 통신과 중간노드들을 최대한 고려한 프로세싱 노드 network상에 효율적으로 분산하는것에 달려있다. 이것을 하기 위하여 가장 좋은 해결책은 중간노드를 최소한 줄인 논리적인 task distribution network와 물리적인 프로세싱 노드 network를 그대로 mapping시키는 것을 생각할 수 있지만 이것은 현실적으로 불가능한 일이다. 그리고 또한 구현된 어떠한 응용 소프트웨어가 실제 현장에서 사용될때에는 개발되어질 때와는 다른 하드웨어에서 실행될것이고 처리되어지는 데이타 또한 다를 것이다. 이러한 경우에, 개발시 아무리 효율적인 task distribution network를 사용하였다 하더라도 그것이 실제 현장에서는 적합하지 않는 경우도 생기는 것은 당연한 사실이다.

본 연구에서는 멀티미디어 데이타 처리 프로그램과 같이 처리되는 데이타 양이 많고 각 Task 사이의 통신이 빈번한 병렬처리 프로그램을 병렬처리 시스템에서 수행하고자 할때 발생하는 Communication Overhead를 최소화 할 수 있도록

Task Distribution Network를 자동 생성하고 Communication BottleNeck을 제거할 수 있는 Adaptive Task Distribution Network Reconfiguration Model을 제안하였다. 이 모델을 여러 Sample Program들에 적용해본 결과 Task에 대한 Processor Allocation 시에 Load Balancing 만을 고려한 경우와 비교하여 볼 때 한 프로그램의 수행 완료를 위해 소요되는 시간을 최대한으로 줄일 수 있었다.

또한 여기서 제시된 모델은 한 병렬처리 프로그램의 개발이 완료된 후에 현장에서 사용될 때 발생하는 하드웨어 구성의 차이와 주로 처리되는 데이타의 유형에 따라 최적의 Task Distribution Network을 자동적으로 구성할 수 있으며, 이러한 과정을 프로그램의 성능의 증가가 Saturation Point에 도달할 때까지 반복하여 수행함으로써 한 Site에서의 Adaptation이 가능하였다.

참 고 문 헌

여 백

참 고 문 헌

[Stone-a] H.S. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," IEEE Trans. Software Eng., vol. SE-3, Jan. 1977.

[Stone-b] H.S. Stone, "Critical load factors in distributed computer systems," IEEE Trans. Software Eng., vol. SE-4, May 1978.

[Bokhari] Shahid H. Bokhari, "Dual Processor Scheduling with Dynamic Reassignment," IEEE Trans. Software Eng., vol. SE-5, July 1979.

[Abraham] Santosh G.. Abraham and Edward S. Davidson, "Task Assignment Using Network Flow Methods for Minimizing Communication In n-Processor Systems," Univ. of Illinois at Urbana-Champaign, Technical Report No. CSRD-598, Sept. 1986.

[Oehlrich] C.W. Oehlrich, "Performance Evaluation of a Communication System for Transputer-Networks Based on Monitored Event Traces," in Proc. 18th Int. Conf. on Computer Architecture, May 1991.

[Patton] Peter C. Patton, "Multiprocessors : Architecture and Applications," IEEE Trans. Computer, June 1985.

[Barron] Barron, I., et al. "The Transputer," Electronics, 17th Electronics, November 1983.

[INMOS] INMOS, "The Transputer Databook," 1988.

[Stevens] Whitby-Stevens, C., "The Transputer," In Proceedings of the 12th Int. Symp. on Computer Architecture, 1985.

[Ackerman] M.E. Hodges, R.M. Sasnett, and M.S. Ackerman, "A Construction Set for Multimedia Applications," IEEE Software, Jan. 1989.

[Lo] Virginia Mary Lo, "Heuristic Algorithms for Task Assignment in Distributed Systems," IEEE Trans. Computer, 1984.

[Enslow] Philip H. Enslow, Jr., "What is a "Distributed Data Processing System?" IEEE Trans. Computer Jan. 1978.

[Gajski] Daniel D. Gajski and Jih-Kwon Peir, "Essential Issues in Multiprocessor Systems," IEEE Trans. Computer, June 1985.

[Ahmad] Ishfaq Ahmad and Arif Ghntoor, "Semi-Distributed Load Balancing For Massively Parallel Multicomputer Systems," IEEE Trans. Software Engineering, Oct. 1991.

[Rao] Gururaj S. Rao and Harold S. Stone, "Assignment of Tasks in a Distributed Processor System with Limited Memory," IEEE Trans. Computers, April 1979.

[PurHof] James M. Purtilo and Christine R. Hofmeister, "Dynamic Reconfiguration of Distributed Programs," in Proc. 11th Int. Conf. on Distributed Computing Systems , May 1991.

[Shivar] Niranjan G. Shivaratri and Mukesh Singhal, "A Transfer Policy for Global Scheduling Algorithms to Schedule Tasks With Deadlines," in Proc. 11th Int. Conf. on Distributed Computing Systems , May , March 1991.

[Chuang] Po-Jen Chuang and Nian-Feng Tzeng, "An Efficient Submesh Allocation Strategy for Mesh Computer Systems," in Proc. 11th Int.

Conf. on Distributed Computing Systems , May , March 1991.

[Helios] Perihelion Software, "The Helios Operating System," Prentice Hall, 1989

[Cld] Perihelion Software, "The CDL Guide," DSL, 1990.

여 백

Appendix

여 백

Appendix

가 : Single-Process Search 프로그램의 수행 결과.

```
-------- Binary Search ----------
*** The word to search is : 대상은
    I Found it by worker [0]. # of Comparisons [10]
        Times used : utime (349) stime (0)
*** The word to search is : 라
    I Found it by worker [0]. # of Comparisons [12]
        Times used : utime (397) stime (0)
*** The word to search is : 레벨의
    I Found it by worker [0]. # of Comparisons [10]
        Times used : utime (338) stime (0)
*** The word to search is : 바와
    I Found it by worker [0]. # of Comparisons [10]
        Times used : utime (314) stime (0)
*** The word to search is : 밸브등의
    I Found it by worker [0]. # of Comparisons [11]
        Times used : utime (355) stime (0)
*** The word to search is : 최꼬
    I CANNOT find [ZllBl]. # of Comparisons [14]
        Times used : utime (334) stime (0)
```

*** The word to search is : 같다고

I Found it by worker [0]. # of Comparisons [8]

Times used : utime (218) stime (0)

*** The word to search is : 대역폭이

I Found it by worker [0]. # of Comparisons [7]

Times used : utime (245) stime (0)

--------- Fibonacci Search ---------

*** The word to search is : 대상은

I Found it by worker [0]. # of Comparisons [10]

Times used : utime (426) stime (0)

*** The word to search is : 라

I Found it by worker [0]. # of Comparisons [12]

Times used : utime (464) stime (0)

*** The word to search is : 레벨의

I Found it by worker [0]. # of Comparisons [10]

Times used : utime (436) stime (0)

*** The word to search is : 바와

I Found it by worker [0]. # of Comparisons [10]

Times used : utime (395) stime (0)

*** The word to search is : 밸브등의

I Found it by worker [0]. # of Comparisons [8]

Times used : utime (355) stime (0)

*** The word to search is : 최꼬

I CANNOT find [ZllBl]. # of Comparisons [14]

Times used : utime (494) stime (0)

*** The word to search is : 같다고

I Found it by worker [0]. # of Comparisons [11]

Times used : utime (468) stime (0)

*** The word to search is : 대역폭이

I Found it by worker [0]. # of Comparisons [11]

Times used : utime (447) stime (0)

--------- Sequential Search ---------

*** The word to search is : 대상은

I Found it by worker [0]. # of Comparisons [267]

Times used : utime (6) stime (0)

*** The word to search is : 라

I Found it by worker [0]]. # of Comparisons [379]

Times used : utime (10) stime (0)

*** The word to search is : 레벨의

I Found it by worker [0]]. # of Comparisons [382]

Times used : utime (11) stime (0)

*** The word to search is : 바와

I Found it by worker [0]]. # of Comparisons [479]

Times used : utime (13) stime (0)

*** The word to search is : 밸브등의

I Found it by worker [0]]. # of Comparisons [480]

Times used : utime (13) stime (0)

*** The word to search is : 최고

I CANNOT find [ZllBl]]. # of Comparisons [1459]

Times used : utime (34) stime (0)

*** The word to search is : 같다고

I Found it by worker [0]]. # of Comparisons [31]

Times used : utime (3) stime (0)

*** The word to search is : 대역폭이

I Found it by worker [0]]. # of Comparisons [269]

Times used : utime (7) stime (0)

--------- Interpolation Search ---------

*** The word to search is : 대상은

I Found it by worker [0]]. # of Comparisons [7]

Times used : utime (540) stime (0)

*** The word to search is : 라

I Found it by worker [0]]. # of Comparisons [5]

Times used : utime (384) stime (0)

*** The word to search is : 레벨의

I Found it by worker [0]]. # of Comparisons [5]

Times used : utime (341) stime (0)

*** The word to search is : 바와

I Found it by worker [0]]. # of Comparisons [9]

Times used : utime (546) stime (0)

*** The word to search is : 밸브등의

I Found it by worker [0]]. # of Comparisons [8]

Times used : utime (443) stime (0)

*** The word to search is : 최고

I CANNOT find [ZlIBl]]. # of Comparisons [5]

Times used : utime (385) stime (0)

*** The word to search is : 같다고

I Found it by worker [0]]. # of Comparisons [9]

Times used : utime (559) stime (0)

*** The word to search is : 대역폭이

I Found it by worker [0]]. # of Comparisons [7]

Times used : utime (563) stime (0)

나 : 병렬 Search 프로그램의 수행 결과

-------- Binary Search ----------

*** The word to search is : 대상은

I Found it by worker [6]]. # of Comparisons [6]

Times used : utime (120) stime (0)

*** The word to search is : 라

I Found it by worker [8]]. # of Comparisons [8]

Times used : utime (211) stime (0)

*** The word to search is : 레벨의

I Found it by worker [1]]. # of Comparisons [6]

Times used : utime (177) stime (0)

*** The word to search is : 바와

I Found it by worker [8]]. # of Comparisons [6]

Times used : utime (190) stime (0)

*** The word to search is : 밸브등의

I Found it by worker [9]]. # of Comparisons [7]

Times used : utime (173) stime (0)

*** The word to search is : 최고

I CANNOT find [ZllBl]]. # of Comparisons [11]

Times used : utime (219) stime (0)

*** The word to search is : 같다고

I Found it by worker [0]]. # of Comparisons [7]

Times used : utime (186) stime (0)

*** The word to search is : 대역폭이

I Found it by worker [8]]. # of Comparisons [4]

Times used : utime (148) stime (0)

--------- Fibonacci Search ---------

*** The word to search is : 대상은

I Found it by worker [6]]. # of Comparisons [8]

Times used : utime (247) stime (0)

*** The word to search is : 라

I Found it by worker [8]]. # of Comparisons [8]

Times used : utime (224) stime (0)

*** The word to search is : 레벨의

I Found it by worker [1]]. # of Comparisons [6]

Times used : utime (151) stime (0)

*** The word to search is : 바와

I Found it by worker [8]]. # of Comparisons [3]

Times used : utime (110) stime (0)

*** The word to search is : 밸브등의

I Found it by worker [9]]. # of Comparisons [4]

Times used : utime (216) stime (0)

*** The word to search is : 최고

I CANNOT find [ZllBl]]. # of Comparisons [9]

Times used : utime (302) stime (0)

*** The word to search is : 같다고

I Found it by worker [0]]. # of Comparisons [8]

Times used : utime (241) stime (0)

*** The word to search is : 대역폭이

I Found it by worker [8]]. # of Comparisons [5]

Times used : utime (181) stime (0)

--------- Sequential Search ---------

*** The word to search is : 대상은

I Found it by worker [6]]. # of Comparisons [27]

Times used : utime (4) stime (0)

*** The word to search is : 라

I Found it by worker [8]]. # of Comparisons [38]

Times used : utime (7) stime (0)

*** The word to search is : 레벨의

I Found it by worker [1]]. # of Comparisons [39]

Times used : utime (4) stime (0)

*** The word to search is : 바와

I Found it by worker [8]]. # of Comparisons [48]

Times used : utime (7) stime (0)

*** The word to search is : 밸브등의

I Found it by worker [9]]. # of Comparisons [48]

Times used : utime (7) stime (0)

*** The word to search is : 최꼬

I CANNOT find [ZllBl]. # of Comparisons [145]

Times used : utime (10) stime (0)

*** The word to search is : 같다고

I Found it by worker [0]]. # of Comparisons [4]

Times used : utime (3) stime (0)

*** The word to search is : 대역폭이

I Found it by worker [8]]. # of Comparisons [27]

Times used : utime (6) stime (0)

--------- Interpolation Search ---------

*** The word to search is : 대상은

I Found it by worker [6]]. # of Comparisons [3]

Times used : utime (349) stime (0)

*** The word to search is : 라

I Found it by worker [8]]. # of Comparisons [3]

Times used : utime (544) stime (0)

*** The word to search is : 레벨의

I Found it by worker [1]]. # of Comparisons [4]

Times used : utime (602) stime (0)

*** The word to search is : 바와

I Found it by worker [8]]. # of Comparisons [6]

Times used : utime (534) stime (0)

*** The word to search is : 밸브등의

I Found it by worker [9]]. # of Comparisons [6]

Times used : utime (641) stime (0)

*** The word to search is : 최고

I CANNOT find [ZlIBI]]. # of Comparisons [4]

Times used : utime (343) stime (0)

*** The word to search is : 같다고

I Found it by worker [0]]. # of Comparisons [3]

Times used : utime (577) stime (0)

*** The word to search is : 대역폭이

I Found it by worker [8]]. # of Comparisons [4]

Times used : utime (618) stime (0).bp

주 의

1. 이 보고서는 과학기술처에서 시행한 특정연구 개발사업의 연구보고서이다.

2. 이 연구개발 내용을 대외적으로 발표할 때에는 반드시 과학기술처에서 시행한 특정연구 개발사업의 연구결과임을 밝혀야 한다.