논문 99-24-3B-24 한국통신학회논문지 '99-3 Vol.24 No.3B

PVM을 이용한 적응형 병렬 스택필터의 구현

정회원 윤 영 록, 변 혜 란*, 유 지 상**

Implementation of Parallel Adaptive Stack Filter Algorithm using PVM

Young-Rock Yoon, Hye-Ran Byun*, Ji-Sang Yoo** *Regular Members*

요 약

스택필터는 신호복원성이 뛰어난 대표적 비선형 필터이다. 그러나 윈도우의 크기가 큰 스택필터를 설계하는 것은 계산량이 많아 시간이 많이 소모되므로 그 응용에 한계가 있었다. 이러한 단점을 극복하기 위해 적응형 병렬 스택필터 알고리즘이 개발되었다. 본 논문에서는 워크스테이션 네트워크 환경을 사용하여 새로운 상이 병렬 처리 환경을 제공하는 소프트웨어인 Parallel Virtual Machine(PVM)을 사용하여 적응형 스택필터 알고리즘을 구현하는 방법을 제안하였다. 본 논문에서 제안한 방법에서는 각 워크스테이션간의 데이터 이동을 최소화하기 위해 적응형 스택필터 알고리즘의 특성을 분석하여 프로그램 실행 초기에 각 워크스테이션에 할당될 작업량을 결정하는 부하 조정 방식을 사용하였다. 또한 PVM은 대역폭이 낮은 기존의 근거리 통신망을 사용하기 때문에 각 워크스테이션간의 메모리 접근 횟수를 최소화 할 수 있도록 메모리를 공유하는 방식을 제안하였다. 이 방법은 사용된 데이터와 윈도우의 크기가 클 때 기존에 한 대의 컴퓨터에서 구현되었던 결과에 비해 영상의 신호 복원력은 같으나 수행시간에서 훨씬 좋은 결과를 보였다.

ABSTRACT

Stack filter was widely used because it showed good performance for signal restoration and noise reduction especially for impulsive noises. Due to the inherent serial nature of the training algorithm, however, it requires too much resource. Parallel adaptive stack filter algorithm which can be implemented in parallel was developed to overcome this problem. In this paper, a new implementation of adaptive stack filter training algorithm was developed using PVM, which is a new heterogeneous parallel computing environment using serial network-available computers. In order to maximize the merit of heterogeneous distributed computing environment and minimize data transfer, load of each host was balanced in early time of process. Because low-bandwidth high-latency network was used, distributed shared memory models were proposed and implemented to minimize the network communication. It shows a better execution time than previous one-host implementation when the data and window sizes are large.

I. 서 론

비선형 필터는 영상 신호 및 음성 신호의 충격성 잡음의 제거에 좋은 효과를 나타내는 것으로 알려져 있다. 많은 비선형 필터들 가운데서 스택필터는 평균 제곱 오차에 의한 최적 알고리즘이 알려져 있기 때문에 이러한 충격성 잡음의 제거에 널리 사용되고 있다. 그러나 스택필터 알고리즘은 잡음에 의해 왜곡된 영상과 원 영상을 이용한 학습 과정이 필요하며, 이러한 학습 과정은 병렬화 되기 힘든 순

* 연세대학교 기계전자공학부 ** 광운대학교 전자공학과(jsyoo@daisy.kwangwoon.ac.kr)
논문번호 : 98289-0715, 접수일자 : 1998년 7월 15일
※ 이 논문은 1998년도 광운대학교 학술연구비에 의하여 연구되었음.

차적 특성을 갖기 때문에 많은 시간과 컴퓨터 자원을 필요로 하였다. 이러한 문제점을 해결하기 위하여 적응형 병렬 스택필터 알고리즘이 개발되었다[1]. 이 알고리즘은 내부적으로 병렬 구조를 가진 컴퓨터에서 구현이 되어 좋은 결과를 보였다.

병렬 처리 분야에서 일반적인 컴퓨팅 환경으로 자리를 잡아가고 있는 워크스테이션 네트워크 환경은 다양한 구조를 가진 워크스테이션들을 일반적인 네트워크로 연결하는 환경으로 메시지 전송방식을 사용하여 병렬 프로그램을 구현할 수 있다. 이 환경은 슈퍼컴퓨터를 보유하고 있지 못한 사용자들에게 기존에 보유하고 있는 자원을 이용하여 높은 계산 능력을 얻어낼 수 있는 기반을 제공한다. 워크스테이션 네트워크 환경은 계산 능력이 서로 다른 컴퓨터들로 구성되어 있기 때문에 각 CPU의 처리속도가 서로 다르며 자원을 요구하는 작업들이 시간이 지남에 따라 변화하기 때문에 특정 작업이 자원을 사용할 수 있는 정도, 워크스테이션의 가용한 처리 능력이 변화한다. 따라서 이러한 환경에서는 실행 시간 동안 가용한 처리능력에 따라서 각각의 워크스테이션에 적절한 양의 작업을 할당하는 부하 조정이 필요하게 된다. 또한 이러한 환경에서는 각 워크스테이션이 일반적인 네트워크를 통한 메시지 전송방식을 통하여 통신을 수행함으로 원거리의 워크스테이션 메모리에 접근하기 위해서는 워크스테이션의 내부 메모리에 접근하는 시간보다 훨씬 많은 시간을 요구하게 된다. 그러므로 구현 될 어플리케이션에서는 각 워크스테이션간의 메모리 접근 횟수를 최소화하는 방식으로 메모리를 공유하여야 한다.

본 논문에서는 적응형 스택필터 알고리즘을 워크스테이션 네트워크 환경에 구현하면서 알고리즘의 특성을 분석하여 각 워크스테이션간의 데이터 이동을 최소화하기 위해 프로그램 실행 초기에 각 워크스테이션에 할당될 작업량을 결정하는 부하 조정 방식을 제안하였으며 각 워크스테이션간의 메모리 접근 횟수를 최소화 할 수 있도록 메모리를 공유하는 방식을 제안하였다.

부하 조정을 위해서는 각 워크스테이션의 CPU처리 능력과 각 워크스테이션이 작업을 시작할 시간에서의 작업 부하, 그리고 워크스테이션간의 통신에 걸리는 네트워크 전송 시간을 측정하여 이러한 요소를 모두 포괄하는 작업 부하 모델을 제시하고 각 워크스테이션에서 이러한 전체 작업 부하를 평균화하도록 하는 최소상승방법(minimal uplifting method)을 사용하였다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 본 논문의 연구 배경으로서 적응형 병렬 스택필터 알고리즘과 상이 병렬 처리 환경에 대하여 소개하고 알고리즘 구현 시 고려해야 할 사항들을 설명하였다. Ⅲ장에서는 구현된 부하 조정 방법과 메모리 공유 방법에 관해 설명하고 Ⅳ장에서는 실험 결과를 기존의 연구 결과와 비교 분석하였다. Ⅴ장에서는 얻어진 결과를 토대로 결론과 향후 앞으로의 연구 계획을 서술하였다.

Ⅱ. 연구 배경

2.1 스택필터 알고리즘

2.1.1 최적 스택필터 알고리즘

스택필터는 임계치분할특성(threshold decomposition property)으로 알려진 약중첩특성(weak superposition property)과 스택킹특성(stacking property)으로 알려진 순서특성(ordering property)을 만족시키는 비선형 필터의 일종이다. 임계치분할 특성은 다음과 같이 정의된다. 일반적으로 0에서 M까지의 흑백 명도를 갖는 흑백 영상 X 는 다음과 같이 이진 값을 갖는 영상들의 합으로 나타내어질 수 있다.

$$X(s) = \sum_{l=1}^{M} x_l(s)$$

이 경우 각 l값에 대하여 이진 영상 $x_l(s)$는 $X(s)$를 l에 관하여 임계치를 적용함으로써 얻을 수 있다. 즉, $x_l(s)$는 다음과 같이 나타내어진다.

$$x_l(s) = \begin{cases} 1, & X(s) \geq l \\ 0, & X(s) < l \end{cases}$$

이제 W를 P의 크기를 갖는 윈도우라고 하고 W(s)를 윈도우 W를 대상이 되는 이진 영상의 s위치에 적용할 때 나타나는 P개의 점의 배열이라고 한다면, 입력 영상 X의 윈도우 배열 $W_X(s)$는 동일하게 다음과 같이 임계치 분할될 수 있다.

$$W_x(s) = \sum_{l=1}^{M} w_{X,l}(s)$$

스택킹특성은 다음과 같이 정의된다. 영상 X를 l에 관하여 임계치 분할한 이진 영상에서 s위치의 윈도우를 $w_{X,l}(s)$라 하자. 어떤 두개의 양의 정수 k와 l에 관하여 $k \leq l$일 때 윈도우 $w_{X,l}(s)$가 어떤 f라는 함수에 적용되었을 때의 출력이 1이면 함수 f에

적용된 $w_{X,k}(s)$의 출력도 1일 경우 함수 f는 스택킹특성을 가진다고 한다. 즉 어떤 두개의 양의 정수 k와 l에 관하여 $k \le l$ 일 때, 반드시 아래의 식이 성립되어야 한다.

$$f(w_{X,k}(s)) \ge f(w_{X,l}(s))$$

어떤 부울함수가 스택킹특성을 만족할 때 이를 정부울함수(positive Boolean function)라고 한다[2].

위의 두 가지 특성을 스택필터에 관하여 다시 정리하면, 정부울함수 $f(\cdot)$와 윈도우 $w_{X,l}$에 대하여 수학적으로 다음과 같은 약중첩특성이 스택필터 S_f에 대해서 성립한다.

$$\begin{aligned} S_f(w_X(s)) &= S_f\left(\sum_{l=1}^{M} w_{X,l}(s)\right) \\ &= \sum_{l=1}^{M} S_f(w_{X,l}(s)) \\ &= \sum_{l=1}^{M} f(w_{X,l}(s)) \end{aligned}$$

즉, 스택필터의 작동은 이진 영상 입력에 적용되는 상응하는 정부울함수의 작동과 같으므로 스택필터를 학습하는 과정은 주어진 오차 기준을 만족하는 정부울함수를 구하는 것과 같다. 최적 스택필터를 구하는 오차기준으로서 필터의 출력과 요구되는 영상간의 평균 절대 오차를 최소화하는 방법을 들 수 있다[3]. 만약 X가 요구되어지는 영상이고, $\overline{X}$가 잡음에 의해 손상된 입력 영상이라고 한다면, 두 영상간의 오차는 다음과 같은 적응 방법에 의해 적절한 f를 얻어냄으로써 최소화시킬 수 있다.

$$\begin{aligned} MAE_f &= E\{|X(s) - S_f(W_{\overline{X}}(s))|\} \\ &= E\left\{\left|\sum_{l=1}^{M}(x_l(s) - f(w_{\overline{X},l}(s)))\right|\right\} \\ &\le \sum_{l=1}^{M} E\{|x_l(s) - f(w_{\overline{X},l}(s))|\} \end{aligned}$$

즉, 위의 수식에서 나타나는 한계 오차를 최소화시킴으로써 입력 영상의 s위치에서의 최적의 판단이 s위치에서 요구되는 값이 1보다 큰지 작은지에 대한 결정이 될 수 있도록 하는 함수 $f(\cdot)$를 찾을 수 있다. 이와 같은 과정을 통해 최적 스택필터 알고리즘은 0-1 정수 선형 프로그램(zero-one integer linear program)으로 표현될 수 있다. 이 프로그램은 NP-complete이지만 필터가 그 결정 값을 임의의 수로 할 수 있도록 제한을 완화시키면 0-1 선형 프로그램은 다음과 같은 선형 프로그램으로 재구성될 수 있다[4].

$$\text{minimize} \quad \sum_{i=1}^{2^p} C_i \cdot d_i$$

subject to the constraints :

$d_i \le d_j$ if $a_i \le a_j$

$0 \le d_j \le 1$ for all j

이 경우 d_i는 필터가 이진수 배열 a_i을 입력받을 때 1을 출력할 확률이며 이렇게 필터를 변경시키는 과정을 약한 결정(soft decision)과정이라 한다. 학습의 마지막 과정에서 필터의 결정 변수를 1/2을 임계치로 하여 1/2보다 크면 1로 1/2보다 작으면 0으로 하는 강한 결정(hard decision)에 의해 부울함수를 얻게 된다.

2.1.2 적응형 병렬 스택필터 알고리즘

최적 스택필터 알고리즘은 윈도우의 크기가 커질수록 부울함수 테이블의 크기가 지수 함수적으로 증가하게 되므로 함수의 스택킹특성을 보장하기 위해서는 막대한 시간을 요구한다. 이러한 문제점을 해결하기 위하여 적응형 스택필터 알고리즘이 개발되었다[5]. 그러나 이 알고리즘에서도 순차적인 알고리즘의 특성은 그대로 남아있어 수행 시간의 개선은 이루어지지 못했다. 그 후에 적응형 스택필터 알고리즘의 순차적인 특징을 해결하기 위하여 적응형 병렬 스택필터 알고리즘이 개발되었다[1]. 새로운 알고리즘은 두 가지 측면에서 알고리즘의 병렬 구현을 가능하게 해 주었다.

첫 번째로, 원래의 알고리즘과는 달리 새로운 알고리즘에서는 매번 부울함수 테이블이 변경될 때마다 스택킹특성이 확인되지 않고 임의의 L번의 변경 후에 확인된다. 이 경우 L을 전체 입력 영상으로 가정하면 부울함수 테이블의 증감 값은 입력 영상 위의 각각의 윈도우 위치에서 필터의 출력에 대해 원하는 값에 대한 오차로 볼 수 있으며, 최적 필터는 학습 과정이 모두 끝난 후에 부울함수를 0 에 대해 임계치 분할을 수행함으로써 구할 수 있다. 이 때 각 이진 영상의 처리는 서로 독립적으로 수행될 수 있으므로 병렬 구현이 가능하다.

두 번째로, 병렬 구현 가능한 스택킹특성 확인 방법이 제안되었다[그림 1]. 이 방법에서는 각 스텝마다 해밍 거리(hamming distance)가 1인 두개의 원소들간의 스택킹특성을 확인함으로써 서로 비 의존

적인 원소들간의 병렬 처리가 가능하도록 하였으며, 이 방법에 의해 처리된 부울함수가 최적 부울함수로 수렴한다는 사실이 [1]에 증명되었다.

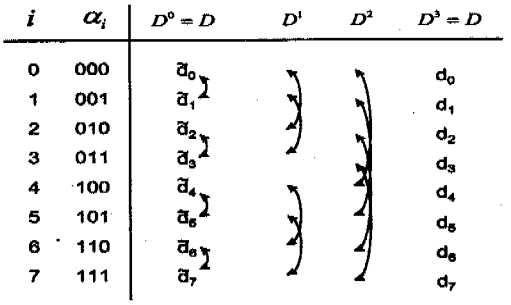


그림 1. 병렬 스택킹특성 확인 방법
Fig. 1 Parallel stacking property enforcing scheme

적응형 스택필터의 학습 알고리즘은 크게 이진 영상 데이터에서 얻어진 결정 벡터에 의해 양성 논리 함수 테이블을 변경하는 과정과 일정 간격마다 양성 논리 함수 테이블이 스택킹특성을 가지도록 스택킹특성을 가하는 두 가지 과정으로 나뉜다. 위에서 언급한 것과 같이 스택킹특성을 가하는 과정의 간격은 임의의 간격으로 정할 수 있으며 대개 이 간격은 주어진 입력 영상 전체의 처리가 이루어지는 간격으로 정한다. 이 과정은 얻어진 양성 논리 함수 테이블이 충분히 최적 논리 함수에 수렴할 때까지 반복된다.

입력 영상 전체의 처리가 이루어진 시점에서의 논리 함수 테이블은 반복되는 과정에서 항상 일정하므로 처음 얻어진 논리 함수 테이블은 임의의 기억 장소에 보관되었다가 스택킹특성이 가하여진 양성 논리 함수 테이블에 더함으로써 입력 영상의 처리 과정을 반복할 필요 없이 알고리즘을 반복 수행할 수 있다.

2.2 워크스테이션 네트워크 환경과 PVM

2.2.1 워크스테이션 네트워크 환경

최근의 컴퓨팅 환경의 특징은 다양한 구조의 워크스테이션들이 비교적 빠른 네트워크에 의해 연결되어 있다는 점이다. 그러므로 만약 각 워크스테이션을 연결하여 병렬 처리 환경을 만들 수 있는 프로그래밍 환경이 제공된다면 이러한 컴퓨팅 환경은 상이병렬처리환경(heterogeneous parallel computing environment)으로 정의될 수 있다.

상이 병렬 처리 환경의 가장 큰 장점은 그 응용 범위가 특정 어플리케이션에 제한되지 않고, 비교적 적은 비용으로 전체 환경의 성능을 향상시켜 줄 수가 있다는 데에 있다[6]. 그러나 이와 같은 상이 병렬 처리 환경의 장점을 살릴 수 있기 위해서는 MPP(massively parallel processor)와 같은 슈퍼컴퓨터 급의 동질병렬처리환경(homogeneous parallel computing environment)과는 다른 각도에서 어플리케이션을 바라보아야 한다. 주어지는 작업량을 고르게 각 프로세서에게 할당하는 MPP와는 달리 상이 병렬 처리 환경에서는 각 프로세서마다 걸리게 되는 작업량이 서로 다르고 각 프로세서의 처리 능력도 서로 다르므로 어플리케이션의 작업 특성을 분석하여 전체 수행 시간을 최소화 할 수 있도록 작업을 분할하고 가장 적합한 컴퓨터로의 매핑과 올바른 작업량의 할당을 고려하여야 한다[7]. 상이 병렬 처리 환경에서의 각 프로세서들은 곧 병렬 환경 구성에 참여한 각 워크스테이션들이다. 본 논문에서는 앞으로 상이 병렬 처리 환경에 참여한 워크스테이션을 프로세서로 언급하겠다.

2.2.2 Parallel Virtual Machine (PVM)

PVM은 프로그래머로 하여금 상이 병렬 환경을 하나의 커다란 가상 병렬 컴퓨터처럼 사용할 수 있도록 하는 병렬 프로그래밍 환경을 제공한다[8,9]. 병렬 환경을 구성하는 각각의 컴퓨터들로는 공유 메모리 또는 지역적인 메모리만을 가지는 멀티 프로세서 컴퓨터, 벡터 슈퍼 컴퓨터, 워크스테이션, 또는 일반적인 PC 등이 모두 가용하며 이들은 여러 종류의 네트워크로 연결되어 있다.

PVM은 상이 환경 내의 컴퓨터를 병렬 처리 환경에 포함시키거나 제거하는 기능 등의 병렬 처리 환경의 운용을 위한 기능들을 제공하는 사용자 인터페이스와 상이 환경에서 병렬 어플리케이션을 개발할 수 있게 해주는 프로그래밍 라이브러리로 구성된다. 병렬 프로그래밍에서는 각 프로세서간에 데이터 통신과 작업 동기화 등을 통해 메모리의 일관성이 유지되어야 하므로 이러한 작업을 가능하게 해주는 방법이 요구된다. PVM은 분산 되어있는 컴퓨터간의 데이터 통신을 위해 상이 환경에서 가장 널리 사용되고 있는 메시지 전송 방식을 사용하는 프로그래밍 함수들을 제공한다.

PVM은 일반적으로 특수한 목적으로 만들어진 CPU나 네트워크가 아닌 일반적인 목적의 네트워크 환경에서 사용된다. 이러한 환경에서는 운영체제의 작동이나 화일 시스템 오버헤드 그리고 네트워크

통신에 걸리는 시간 등을 예측하기가 힘들다. 그러므로 주어진 어플리케이션의 순수한 수행능력을 측정하기가 매우 힘들며 따라서 최적의 수행 능력을 얻는 것이 어렵다.

Ⅲ. 적응형 스택필터의 병렬 구현

적응형 스택필터의 학습 과정은 입력 영상 처리 과정과 스택킹특성 확인 과정으로 나뉠 수 있다. 우리가 사용하는 워크스테이션 환경에서는 네트워크의 대역폭(bandwidth)이 작고 잠복 시간(latency)이 크므로 네트워크 통신의 양과 횟수를 최대한 줄여야 한다. 학습의 결과로 생성되는 양성 부울함수의 크기는 적용되는 윈도우의 크기에 지수 함수적으로 증가하기 때문에 부울함수 테이블의 전송에 걸리는 시간은 각 프로세서의 수행 시간에 비해 매우 클 수가 있다.

3.1 마스터/슬레이브 방식

구현될 필터의 학습과정은 두개의 부분으로 나뉜다. 마스터는 각 슬레이브들을 초기화시키고, 슬레이브 들의 벤치마크에 필요한 정보를 PVM 구성 정보를 통하여 얻은 후에 각 프로세서에 알맞는 부하의 양을 결정하고 할당한다. 슬레이브로 표현되는 각 프로세서들은 수행에 필요한 입력 데이터와 작업량을 마스터로부터 받아 임계치 분할을 수행한 후 생성된 이진 영상 데이터를 이용하여 스택킹특성이 가해지지 않은 부울함수를 만들어 마스터에게 전송한다. 마스터는 전송된 부울함수를 모아서 스택킹특성을 가한다. 마스터 슬레이브 방식은 전체 학습 과정이 모두 병렬로 구현되지 않았다는 단점이 있는 반면 네트워크를 통한 데이터의 전송을 최소화 할 수 있다는 점과 수행에 필요한 정보를 한곳으로 모아서 일관성 있게 처리할 수 있다는 장점이 있다. 구현된 모델은 그림 2에 나타내었다.

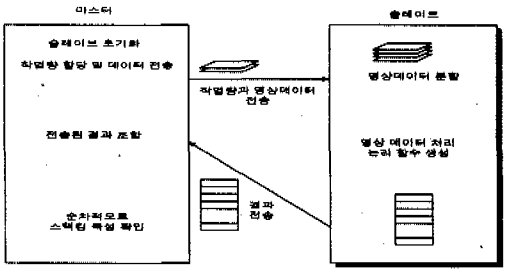


그림 2. 구현된 마스터 슬레이브 모델
Fig. 2 Implemented Master-Slave model

3.2 부하 조정 방법

상이 병렬 처리 환경에서는 프로그램의 전체 수행 시간은 가장 느린 프로세서의 수행 시간에 의존하게 된다. 따라서 각 프로세서의 특성 및 수행 능력을 정확히 파악하여 수행 시간을 예측할 수 있다면 각 프로세서에게 적당한 작업량을 할당함으로써 전체 프로그램의 수행 시간을 단축시킬 수 있다.
부하 조정 방법에는 정적인 부하 조정 방법과 동적인 부하 조정 방법의 두 가지가 있다[10,11,12]. 가장 간단한 정적 부하 조정은 각 프로세서에 할당될 작업량을 어플리케이션이 수행되기 전에 미리 결정을 하는 것으로써 프로그램 수행 중에 부하를 이동시켜야 하는 오버헤드가 발생하지 않으므로 수행되어야 할 작업의 양이 적거나 저속의 통신망에서 수행되어 통신 오버헤드가 많이 발생하는 경우에 적합하다. 동적 부하 조정 방법은 각 프로세서에 할당될 작업량이 어플리케이션이 수행되는 시점에서 그때 그때의 상황에 맞게 할당되는 것으로써 최적의 부하 균형을 얻을 수 있다는 장점이 있는 반면 프로그램 수행 중에 부하의 조정을 위한 프로세서간의 통신이 필요하다는 단점이 있다. 부하조정 방법의 선택은 부하 조정에 걸리는 오버 헤드가 전체 수행 시간에 막대한 영향을 줄 우려가 있으므로 수행할 어플리케이션의 특성을 잘 고려하여 신중히 결정해야 할 사항이다.

각 프로세서의 처리 능력이 서로 다르고 특정 시간에 각 프로세서가 수행해야 할 부하의 양이 다양하게 변화하는 상이 병렬 처리 환경에서는 동적 부하 조정 방법이 적합하겠지만 본 논문에 사용된 워크스테이션 네트워크 환경의 경우 어플리케이션이 수행해야 할 전체 부하의 양이 그리 크지 않고 데이터 전송을 위한 네트워크의 속도가 느린 점을 감안한다면 동적 부하 재 할당 방법은 그 효과보다는 수행에 따른 오버헤드가 더 많을 것이다. 실제로 동적 부하 조정 방법 중 작업 풀을 이용한 방법을 시도 해 본 결과 좋지 않은 결과를 보였다.

부하 조정을 위하여 고려해야 할 사항은 어플리케이션의 특성 외에도 작업이 수행될 환경의 특성이 있다. 이것은 일반적으로 벤치마크라고 하는데 상이 병렬 처리 환경에서 중요한 벤치마크로는 각 프로세서의 수행 능력, 프로그램이 수행 될 상황에서의 각 프로세서의 외부 작업 부하, 그리고 프로세서간의 통신에 걸리는 네트워크 부하 등이 있다. 벤치마크에는 각 프로세서의 수행 능력과 같이 한번

구한 후에는 변화하지 않는 부분과 외부 부하 같이 시간에 따라 변화하는 부분이 있다. 결국 최적의 부하 조정을 위해서는 최대한 많은 벤치마크를 구하는 것이 중요하겠지만 벤치마크를 구하는 작업 자체도 부하 조정과 마찬가지로 오버 헤드를 일으키므로 전체 수행 효율과 효과적인 부하 조정 사이에서 타협점을 찾아내는 것이 중요하다. 본 논문에서는 각 프로세서의 수행 능력을 나타내는 CPU속도, 어느 시점에서의 프로세서의 부하, 그리고 그 프로세서까지의 데이터 전송에 필요한 네트워크 전송 시간을 벤치마크로 사용하였다.

본 논문에서는 슬레이브를 초기화하는 시점에서 네트워크 부하와 각 프로세서가 수행하고 있는 부하의 양을 측정하여 할당될 작업의 양에 반영하는 정적 부하 조정 방법의 일종인 MUM(minimal uplifting method)방법을 사용하였다. 전체 프로세서 중 i 번째 프로세서에 대하여 S_i를 CPU 속도, W_i를 각 프로세서의 부하, N_i를 데이터 전송에 걸리는 시간, 그리고 d_i를 할당될 작업량으로 본다면 모든 벤치마크를 고려한 전체 작업 부하 δ_i를 다음과 같이 정의할 수 있다.:

$$\delta_i = \frac{S_f}{W_i + N_i + d_i} \quad (0 \le i \le m-1,\ m\text{은 프로세서의 수})$$

$$\sum_{i=0}^{m-1} d_i = N \qquad (\text{N 은 수행될 작업의 수})$$

일단 W_i와 N_i을 구한 후에 δ_i값이 모든 프로세서에서 균등하도록 d_i값을 적절히 조절함으로써 적절한 양의 부하를 각 프로세서에게 할당 할 수 있다. $d_i^{(k)}$를 k번의 재분배 후에 할당된 작업량이라고 하면 $d_i^{(0)}$는 모두 동일한 량의 작업량으로 초기화 할 수 있다. 어느 순간에 작업 할당량을 조절하기 위하여 δ_i의 최대값과 최소값을 다음과 같이 정의하며 이것은 매 순간마다 쉽게 구해질 수 있다.

$$\min_{i=0}^{m-1} \delta_i = \delta_{\min}$$

$$\max_{i=0}^{m-1} = \delta_{\max}$$

일단 $\delta_{\min}^{(k)}$와 $\delta_{\max}^{(k)}$를 구하면 $d_{\min}^{(k)}$의 값을 올려주고 $d_{\max}^{(k)}$의 값을 낮춰줌으로써 k+1번째 단계에서 δ의 범위를 좁힐 수 있다. 즉 다음과 같은 식에 의하여 k+1번째의 재분배를 수행할 수 있다.

$$\delta_{\min(k)}^{(k+1)} = \frac{S_{\min(k)}}{W_{\min(k)} + N_{\min(k)} + (d_{\min(k)} - 1)}$$

$$\delta_{\max(k)}^{(k+1)} = \frac{S_{\max(k)}}{W_{\max(k)} + N_{\max(k)} + (d_{\max(k)} + 1)}$$

여기서 $\min(k)$는 δ_i의 최소값이 할당된 프로세서의 번호 $\max(k)$는 δ_i의 최대값이 할당된 프로세서의 번호이다. 마지막으로 $\delta_{\min}$와 $\delta_{\max}$의 범위가 충분히 좁아질 때까지 이러한 단계를 반복함으로써 적절한 작업량을 할당할 수 있게 된다.

Ⅳ. 실험 및 고찰

4.1 실험 환경 및 방법

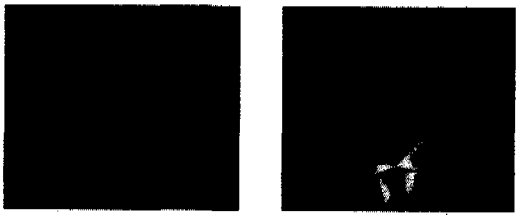
구현 된 알고리즘은 각각 4대와 8대의 워크스테이션에서 수행되었다. 4대는 각각 166 MHz와 200MHz의 클럭 속도를 갖는 Pentium 프로세서를 탑재한 PC에서 LINUX 운영체제를 사용하였고 8대의 경우는 위와 같은 PC환경과 함께 Solaris 운영체제를 사용하는 두 대의 Sun Ultra Sparc 워크 스테이션과 SunOS운영체제를 사용하는 한대의 Sun Sparc 10 워크스테이션을 사용하였다. 네트워크는 흔히 쓰이고 있는 TCP/IP를 사용한 16 port Ethernet HUB/Repeater로 연결된 네트워크를 사용하였다. 각 워크스테이션의 네트워크 인터페이스는 PC의 경우 NE2000 호환기종, SUN 워크스테이션의 경우 워크스테이션과 함께 제공되는 네트워크 인터페이스를 그대로 사용하였다. 사용된 병렬구조 환경은 표 1 에 자세히 수록하였다.

표 1. 실험 환경
Table 1. Experimental environment

이름	CPU 및 처리속도	RAM 용량	운영체제
aihorn	Pentium Pro 200MHz	64 Mb	LINUX
aiokarina	Pentium 166MHz	32 Mb	LINUX
aidrum	Pentium Pro 200MHz	64 Mb	LINUX
torpido	Pentium 166MHz	64 Mb	LINUX
aicleo	Pentium Pro 200MHz	64 Mb	LINUX
aiguitar	Sun Ultra Sparc	64 Mb	Solaris 1.5
emerald	Sun Ultra Sparc	64 Mb	Solaris 2.0
aicello	Sun Sparc 10	32 Mb	SunOS 4.1.3

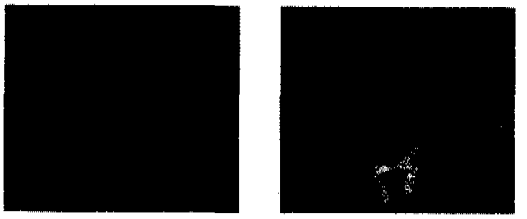
실험에 사용된 데이터는 각각 256×256 크기의 256단계 흑백 명도를 갖는 영상 데이터(Aerial)와 512×512 크기의 256단계 흑백 명도를 갖는 영상

데이터(Albert)를 요구되는 영상으로 사용하였고 같은 영상을 각각 10%의 충격성 잡음으로 왜곡시킨 영상 데이터(AerialI10, AlbertI10)를 왜곡된 영상으로 사용하였다. 사용된 데이터는 그림 3과 4에 수록하였다. 필터의 학습은 윈도우 크기를 각각 3×3과 4×4로 변화시키면서 수행 시간을 측정하였으며 학습에 사용된 영상을 필터링 하여 평균절대오차(mean absolute error)를 측정하였다.



(가) Aerial, 256×256 화소 (나) Albert, 512×512화소

그림 3. 실험에 사용된 영상(요구되는 영상)
Fig 3 Images used in experiments (original images)



(가) AerialI10, 10% 충격성 잡음 (나) AlbertI10, 10% 충격성 잡음

그림 4. 실험에 사용된 영상 (잡음에 의해 왜곡된 영상)
Fig 4. Images used in experiments (noise corrupted images)

4.2 실험 결과 및 분석

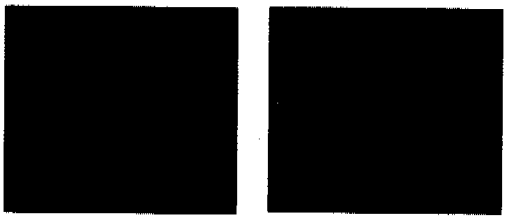
수행된 실험 결과를 기존의 MPP에서 수행되었던 결과와 비교하기 위하여 학습 수행 시간과 필터링 결과의 평균 절대 오차를 [1]에 수록되어 있는 MasPar MP-1[13]에서 수행된 결과와의 비교를 표 2에 보였다.

표 2. 실험환경에 따른 수행시간 및 평균 절대 오차
Table 2. Experimental result

	사용된 영상	Aerial (256×256)		Albert (512×512)	
	윈도우	3×3	4×4	3×3	4×4
수행 시간 (초)	순차방법	22.55	34.80	89.53	130.93
	MasPar MP-1	0.42	18.6	0.83	70.4
	PVM(4 host)	3.79	10.89	23.98	26.44
	PVM(8 host)	3.35	10.42	15.28	24.41
평균절대오차		2.924	2.695	3.223	2.704

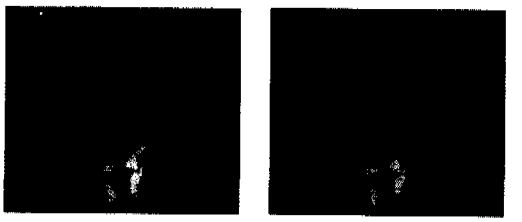
결과에서 보듯이 윈도우의 크기와 입력 영상의 크기가 클 경우 더 좋은 결과를 보였지만 윈도우의 크기와 입력 영상의 크기가 작을 경우 좋지 않은 결과를 보였다. 이것은 입력 데이터의 크기가 큰 경우와 윈도우의 크기가 큰 경우에 각 프로세서간의 통신에 사용되는 오버헤드의 양이 실제 각 프로세서에서 수행되는 작업의 수행 시간에 비해 작아지기 때문이다. 즉 수행되어야 할 작업의 양이 증가할 경우 각 프로세서의 수행 시간의 증가량이 네트워크 부하의 증가량보다 훨씬 크기 때문에 일어나는 현상이다.

그림 5와 6에 학습된 필터를 이용한 필터링 결과를 나타내었다.



(가) 3×3 윈도우 (나) 4×4 윈도우

그림 5. 실험에 사용된 영상의 필터링 결과 (AerialI10)
Fig 5. Filtering results (AerialI10)



(가) 3×3 윈도우 (나) 4×4 윈도우

그림 6. 실험에 사용된 영상의 필터링 결과 (AlbertI10)
Fig 6. Filtering results (AlbertI10)

표 3 에는 각 워크스테이션들의 평균 벤치마크와, 평균 작업 할당량, 그리고 평균 수행 시간이 수록되어 있다. 동적 재 할당이 일어나지 않기 때문에 각 프로세서의 총 수행 시간에는 오차가 나타나지만 실시간의 정확한 부하 예측이 불가능 한 점을 미루어 본다면 결과는 비교적 고른 수행 시간 분포를 나타냄을 알 수 있다. 이 결과에서는 각 프로세서의 수행 시간 오차가 데이터의 크기의 증가에 무관하게 나타남을 알 수 있다.

표 3. 워크스테이션의 평균 벤치마크, 평균 작업 할당량, 평균 수행시간(초)
Table 3. Average benchmark, workload, performance of each workstation(sec)

호스트 이름	평균 부하	평균 네트워크 부하	CPU 속도	평균 작업량	평균 수행시간
aihorn	1.54	0.40	1.0	46.25	3.74
aiokarina	1.49	0.79	0.7	26.50	4.53
aidrum	1.42	0.86	1.0	45.25	3.49
tornido	1.54	1.00	0.7	25.00	5.39
aicleo	1.53	0.97	1.0	43.75	3.44
aiguitar	1.11	0.87	0.7	29.75	3.88
emerald	1.08	0.80	0.7	30.25	4.37
aicello	2.60	0.80	0.6	8.75	3.25

V. 결 론

본 논문에서는 적응형 스택필터 알고리즘을 워크스테이션 네트워크 환경에서 효율적으로 구현하기 위하여 고려할 점들을 제안하고, 두 가지의 메모리 공유 방법에 대해 필터를 구현하였다. 상이 병렬 환경의 장점을 최대한 살리기 위하여 부하 조정 방법을 사용하였지만 부하 조정 시의 오버헤드를 최소화하기 위하여 정적 부하 조정 방법을 택하였다. 실험 결과는 입력 데이터의 크기가 크고 처리되어야 할 작업의 양이 상대적으로 많은 경우에 기존의 연구보다 좋은 결과를 보였는데 그 이유는 프로세서 간의 데이터 전송이 느린 네트워크를 통해서 이루어 져야 하므로 각 프로세서가 담당해야 할 작업의 양이 네트워크 오버헤드보다 상대적으로 큰 경우에 최적의 프로세서 활용도를 보이기 때문이다. 그러나 네트워크와 프로세서간의 불균형이라는 컴퓨팅 환경의 단점에도 불구하고 기존의 컴퓨팅 환경에서 특별한 하드웨어의 도움 없이 병렬 처리 어플리케이션을 합리적인 성능으로 수행할 수 있다는 데에 큰 의의가 있다. 실험 결과는 비교적 적은 수의 프로세서를 사용한 경우에도 기존의 결과보다 좋은 성능을 보였다.

참 고 문 헌

[1] J. Yoo, K. L. Fong, E. J. Coyle, and G. B. Adams III., "Fast Highly Parallel Algorithm for Designing Stack Filters," Annual Allerton Conference on Communication Control and Computing, Monticello, IL, 1993

[2] P.D. Welndt, E. J. Coyle, and N. C. Gallagher Jr., "Stack filters," IEEE Trans. on Acoustics, Speech, and Signal Processing,, Vol. ASSP- 34, No. 4, pp. 898-991, 1986

[3] E. J. Coyle, J-H Lin., and M. Gabbouj, "Optimal stack filtering and the estimation and structural approaches to image processing," IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. 37, pp. 2037-2066, 1989

[4] C. Papadimitriou and K. Steiglitz, "Combinational optimization: algorithm and complexity," Prentice Hall, Engwood Cliffs, NJ, 1987

[5] J-H Lin, T. M. Selke, and E. J. Coyle, "Adaptive stack filtering under the mean absolute error criterion," IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. 38, pp. 938-954, 1989

[6] P. Steenkiste, "Network-Based Multicomputers; A Practical Supercomputer Architecture," IEEE Transactions on Parallel and Distributed Systems, Vol. 7, No. 8, pp. 861-875, 1996

[7] M. Tan, H. J. Siegel, J. K. Antonio, and T. A. Li, "Minimizing the application execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system," IEEE Transactions on Parallel Distributed Systems, Vol. 8, No. 8, pp. 857-871, 1997

[8] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, "PVM: Parallel Virtual Machine. A user's guide and tutorial for networked parallel computing," MIT Press., 1994

[9] V. Sunderam, J. Dongarra, A. Geist, and R. Manchek, "The PVM concurrent computing system: evolution, experiences, and trends," Parallel Computing, Vol. 20, No. 4, pp. 531-547, 1968

[10] D. M. Nicol and P. F. Reynolds Jr., "Optimal dynamic remapping of data parallel computations," IEEE Transactions on Computers, Vol. 39, No. 2, pp. 206-219, Feb. 1990

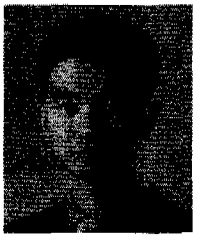
[11] R. V. Hanxleden and L. R. Scott, "Load balancing on message passing architectures,"

Journal of Parallel and Distributed Computing, Vol. 13, pp. 312-324, 1991

[12] M. Huber and T. Schnekenburger, "Heterogeneous partitioning in a workstation network," Heterogeneous Computing Workshop, pp72-77

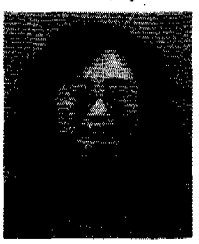
[13] T. Blank, "The MasPar MP-1 Architecture," IEEE Compcon, pp. 20-24, 1990

윤 영 록 (Young-Rock Yoon)



1996년 2월 : 연세대학교 컴퓨터과학과 졸업(이학사)
1998년 2월 : 연세대학교 대학원 컴퓨터과학과 졸업 (공학석사)
1998년 6월~현재 : Purdue 대학교 전기공학과 박사과정
<주관심분야> 신호 및 영상처리, 데이터압축, 컴퓨터 네트워크

변 혜 란 (Hye-Ran Byun)



1980년 2월 : 연세대학교 수학과 졸업 (이학사)
1983년 2월 : 연세대학교 대학원 수학과 졸업 (이학석사)
1993년 12월 : Purdue 대학교 컴퓨터과학과 졸업 (Ph.D.)
1994년 3월~1995년 2월 : 한림대학교 정보공학과 조교수
1995년 3월~현재 : 연세대학교 기계전자공학부 정보산업전공 부교수
<주관심분야> 지능정보처리, 신경망, 영상인식

유 지 상 (Ji-Sang Yoo) 정회원

1985년 2월 : 서울대학교 전자공학과 졸업(공학사)
1987년 2월 : 서울대학교 대학원 전자공학과 졸업 (공학석사)
1993년 5월 : Purdue 대학교 전기공학과 졸업(Ph.D.)
1993년 9월~1994년 8월 : 현대전자산업(주) 산전연구소 선임연구원
1994년 9월~1997년 8월 : 한림대학교 전자공학과 조교수
1997년 9월~현재 : 광운대학교 전자공학과 조교수
<주관심분야> 웨이블릿기반 영상처리, 영상압축, 영상인식, 비선형 신호처리