객체 인식을 위한 다중처리 마이크로프로세서의 성능 평가

(Performance Evaluation of an On-Chip Multiprocessor for Object Recognition)

정 용 화 † 박 경 † 최 성 훈 †† 한 우 종 ††

(Yongwha Chung) (Kyoung Park) (Sung-Hoon Choi) (Woo-Jong Hahn)

요 약 객체 인식은 고성능 컴퓨팅을 필요로 하는 흥미있는 응용 분야이다. 현재 대부분의 고성능 컴퓨터는 슈퍼스칼라 구조의 범용 마이크로프로세서를 채택하고 있으나, 반도체 집적도가 증가함에 따라 슈퍼스칼라 구조를 대신할 다중처리 마이크로프로세서 구조가 제안되고 있다. 본 논문에서는 다중처리 마이크로프로세서 구조가 객체 인식 응용에 적합한지를 분석한다. 성능 특성을 확인하기 위하여 먼저 프로그램 구동방식의 마이크로프로세서 시뮬레이터와 프로그래밍 환경을 개발하였다. 이를 기반으로 시뮬레이션을 수행한 결과, 다중처리 마이크로프로세서가 작은 오버헤드로 쓰레드 수준의 병렬성을 적절히 활용하고 있어 객체 인식 응용에 적합한 구조임을 확인하였다.

Abstract Object recognition is a challenging application for high-performance computing. Currently, the superscalar architecture dominates todays microprocessor marketplace. As more transistors are integrated onto larger die, however, an on-chip multiprocessor is regarded as a promising alternative to the superscalar microprocessor. This paper examines the behavior of the object recognition on the on-chip multiprocessor, which will be employed in general-purpose parallel machines. To obtain the performance characteristics of the microprocessor, a program-driven simulator and its programming environment were developed. The simulation results showed that the on-chip multiprocessor can exploit thread level parallelisms effectively and offer a promising architecture for the object recognition application.

1. 서 론

이미지로부터 관심 있는 객체를 인식하는 프로세스로 정의되는 **객체 인식(object recognition)**은 지난 30년간 컴퓨터 비전 분야의 꾸준한 연구로 공업용 검사(industrial inspection), 자동 운행(vehicle navigation), 항공사진 해석(aerial photo interpretation), 자동 목표물 인식(automatic target recognition) 등의 여러 응용에서 좋은 결과를 보여주고 있다. 그런데 이러한 객체 인식 태스크는 일반적으로 매우 많은 양의 계산을 필요로 한다. 예를 들어, 512×512 크기의 픽셀을 갖는 일련의 컬러(3 바이트/픽셀) 이미지가 표준 프레임율(30 프레임/초)로 입력되는 경우를 가정해보자. 이 경우 초당 24Mbyte의 데이타를 처리해야 하는데, 간단한 특징점 추출 알고리즘이 픽셀 당 수천번의 오퍼레이션을 필요로 하며 일반적인 비전 시스템의 경우는 더욱 복잡한 계산을 필요로 한다. 따라서 이러한 문제를 해결하기 위해서는 고성능 처리(high performance computing)가 필수적임을 알 수 있다[1].

이러한 이유로 미국의 HPCC(High Performance Computing and Communication) 프로젝트에서는 컴퓨터 비전을 Grand Challenge 문제 중의 하나로 정의하였는데[2], 이러한 Grand Challenge 문제를 해결하기

† 정 회 원 : 한국전자통신연구원 컴퓨터시스템연구부 연구원
yongwha@computer.etri.re.kr
kyoung@computer.etri.re.kr
†† 비 회 원 : 한국전자통신연구원 컴퓨터시스템연구부 연구원
shchoi@chestnut.etri.re.kr
wjhan@computer.etri.re.kr
논문접수 : 1999년 10월 16일
심사완료 : 2000년 4월 6일

위해서는 100~1,000 billion operations/second 정도의 계산 능력을 필요로 한다. 그러나 컴퓨터 비전의 계산 특징은 다른 과학계산용 응용들과 매우 다르다. 예를 들어, 3,000×3,000×11 마일의 영역에 대한 일기예보의 경우에 100G 개의 데이타, 1000T 번의 오퍼레이션을 필요로 하며, 이를 1GFLOPS급 Cray에서 수행할 경우 약 280시간이 소요된다. 그러나 일반적인 비전 시스템의 경우, 1K×1K 크기의 픽셀 이미지에 대해서 10K~1M 개의 데이타를 처리해야 하는데, 이는 SUN 워크스테이션에서 약 한시간 정도의 계산을 필요로 한다. 즉, 일기예보의 경우 몇 일 걸리는 계산을 병렬처리에 의해 몇 시간으로(**batch**) 단축시키는 것인데 반하여, 비전 시스템의 경우 몇 시간 또는 몇 십분 걸리는 계산을 병렬처리로 몇 초 이내로(**interactive** 또는 **realtime**) 단축시키는 것을 그 목적으로 한다. 따라서 컴퓨터 비전에서의 성능 요구사항을 만족시키기 위해서는 다른 과학계산용 응용을 병렬화 할 때 무시될 수 있었던 요소들이 컴퓨터 비전을 병렬화 할 때는 매우 조심스럽게 고려되어야만 한다[3~6].

현재 대부분의 고성능 컴퓨터는 *Intel Pentium, Compaq Alpha21164, IBM PowerPC620, Sun UltraSparc, HP PA8000, MIPS R10000* 등 **슈퍼스칼라**(superscalar) 구조[7,8]의 범용 마이크로프로세서를 채택하고 있다. 이러한 슈퍼스칼라 구조는 **명령어 수준의 병렬성(Instruction-Level Parallelism, ILP)**을 활용하여 하나의 싸이클에 여러 개의 명령어를 수행할 수 있으나, 일반적으로 순차 프로그램내의 ILP가 크지 않은 제약에 대한 성능상의 한계를 갖고 있다[9]. 특히, 이렇게 복잡한 마이크로프로세서를 설계하는데 엄청난 비용이 든다는 사실은 슈퍼스칼라 구조를 대신할 새로운 마이크로프로세서 구조에 대한 연구를 가속화 시키고 있다[10~15]. 이중 **다중처리 마이크로프로세서(on-chip multiprocessor microprocessor)**[12~14]는 ILP의 한계를 극복함과 동시에 구현의 용이성을 제공한다는 점에서 차세대 마이크로프로세서 구조로 급부상하고 있다. 즉, ILP 외에 **쓰레드 수준의 병렬성(Thread-Level Parallelism, TLP)**[10~15]을 추가로 제공하고, 반도체 집적도가 증가함에 따라 간단한 구조의 프로세서 유니트를 여러 개 장착함으로써 슈퍼스칼라 구조의 단점을 보완할 수 있다. 그러나 이러한 다중처리 마이크로프로세서가 과학계산 응용[16]에서는 우수한 성능을 나타내는 것으로 발표되고 있지만[12~14], 아직까지 컴퓨터 비전 응용에 적용시킨 결과는 발표되지 않고 있다. 따라서 본 논문에서는 새로운 마이크로프로세서 구조로 급부상하고 있는 다중처리 마이크로프로세서 구조가 일반적인 과학계산 응용과 다른 계산 특성을 갖는 객체 인식 응용에 적합한지를 분석한다.

다중처리 마이크로프로세서를 이용한 객체 인식의 성능 특성을 분석하기 위하여 먼저 프로그램 구동 방식의 전용 아키텍춰 시뮬레이터로서 RapSim을 개발하였다. RapSim은 명령어 시뮬레이터인 **전처리기**(Pre-Processing Unit)와 각 프로세서 모델에 대한 성능 시뮬레이터인 **후처리기**(Post-Processing Unit)로 구성된다. 또한, TLP 제공을 위한 멀티쓰레드 프로그래밍 환경을 개발하였다. 이를 기반으로 시뮬레이션을 수행한 결과, 다중처리 마이크로프로세서가 작은 오버헤드로 TLP를 적절히 활용하고 있어 객체 인식 응용에 우수한 성능을 제공할 수 있다는 것을 확인하였다.

2. 객체 인식 응용과 다중처리 마이크로프로세서

2.1 객체 인식 응용

일반적으로 비전 시스템을 구성하는 태스크는 계산의 특성상 하위레벨(low-level), 중간 레벨(intermediate-level), 상위 레벨(high-level)의 세가지 레벨로 구분된다. 하위 레벨에서 특징점(feature) 추출은 이미지로부터 에지(edge)나 코너(corner) 등과 같은 기본적인 특징점을 추출해 낸다. 그러나 센서 데이타는 일반적으로 잡음이나 계량화에 따른 에러를 수반하기 때문에, 추출된 특징점이 완벽하다고 간주할 수 없다. 이러한 어려움을 해결하기 위해 대부분의 비전 시스템은 hypothesize & verify 방법을 취한다. 즉, 여러 개의 코너로부터 직사각형의 가정(hypothesis)을 생성해 내고, 상위 레벨 분석에 의하여 그러한 가정들을 선택하고 검증하여 최종적으로 객체를 인식하게 된다.

본 논문에서는 이러한 비전 시스템의 예로, 컴퓨터 비전 응용을 수행할 때 병렬 시스템의 성능을 비교하는데 많이 이용되는 DARPA Image Understanding 벤치마크[17]에서 정의한 객체 인식 시스템을 설정하였다. DARPA Image Understanding 벤치마크는 밝기(intensity) 및 깊이(depth) 센서로부터 생성된 이미지가 주어졌을 때 2 1/2 차원의 모빌(mobile) 구조를 인식하는 태스크들로 구성되어 있으며, 다음과 같은 특징을 갖는다.

- convolution, thresholding, connected component labeling, edge tracking, median filter, Hough transform, convex hull, corner

detection 등의 하위 레벨 동작을 포함한다.

- 중간 레벨과 상위 레벨 처리의 예로써 각각 grouping 동작과 graph match를 포함한다.
- occlusion의 존재를 가정하기 때문에, 부분 매치를 고려한다.
- 완전한 해석을 위해서는 두개의 센서로부터 추출한 정보의 활용이 필요하다.
- 정수와 부동소수점 표현의 사용을 동시에 요구한다.
- symbolic과 numeric 데이타 처리 사이에 원활한 데이타 이동을 포함한다.
- bottom-up(data-directed)와 top-down(knowledge 또는 model-directed) 처리를 동시에 요구한다. 이때 top-down 처리에서는 추가적인 특징점 추출을 위해 하위 또는 중간 레벨 데이타 처리를 포함하기도 하고, 전체 계산량을 줄이기 위해 하위 또는 중간 레벨 처리의 제어를 포함하기도 한다.

벤치마크에서 인식해야 할 모빌 구조는 여러 가지 크기, 밝기, 방위, 깊이를 가지는 2차원 직사각형의 집합으로 구성되며, 각각의 직사각형은 Z축(시선 축)에 직각을 이루면서 상수 깊이의 표면을 갖는다. 따라서 각각의 직사각형은 따로 깊이 정보를 갖지 않으며, 깊이는 직사각형간의 공간적 관계에 기인한다(이러한 이유로 모빌 구조가 2 1/2 차원을 갖는다고 함). 또한 장면(scene) 내의 간섭체(clutter)는 또 다른 직사각형으로 구성되는데, 모빌 구조와 유사한 크기, 밝기, 방위, 깊이를 갖는다. 직사각형은 다른 직사각형에 의하여 부분적 또는 전체적으로 가리워(occlude) 질 수 있는데, 하나의 직사각형은 바로 뒤에서 동일한 밝기 또는 약간 큰 깊이를 갖는 또 다른 직사각형이 존재할 때 완전히 사라질 수 있다.

모빌 인식은 인식해야 할 모빌과 유사한 구조들을 나타내는 모델들이 입력으로 주어지면, 이중 주어진 장면과 가장 일치하는 모델을 찾아내는 것으로, 이때 모델은 직사각형의 크기, 방위, 깊이, 공간적 위치 등의 차이를 허용하는 대략의 표현이다. 또한 이러한 모델은 트리 구조를 갖는데, 여기서 링크는 모빌 구조의 링크를 의미하고, 노드는 한 직사각형의 크기, 방위, 깊이, 밝기에 대한 정보를 가지고 있다. 따라서 한 노드의 자손을 연결하는 링크는 그 노드와 바로 밑에 있는 노드의 공간적 위치를 묘사한다.

계산은 밝기와 깊이 이미지에 대한 하위 레벨 비전 처리부터 시작하여, 후보 직사각형을 추출하기 위해 밝기 데이타에 대한 분류(grouping) 처리를 수행한다. 그리고 후보 직사각형들은 저장된 모델과의 부분 매치를 생성하는데 사용된다. 이때 각각의 모델에 대하여 여러 개의 가상 포즈(pose)가 생성될 수 있으며, 각각의 모델 포즈에 대하여 밝기와 깊이 이미지를 탐침(probe)하기 위하여 저장된 정보를 이용한다. 여기서 각각의 탐침은 이미지의 주어진 위치에 직사각형이 존재하는지를 시험하기 위함이다. 직사각형이 실제로 존재하지 않는다는 확실한 증거가 있을 때 그 직사각형의 가정을 기각하는데, 이는 해당 모델 포즈를 삭제하는 결과를 낳는다. 반면 가정의 확인은 그 직사각형에 대한 매치의 세기를 계산하고, 새로운 크기, 방위, 위치 정보로 그 모델 포즈의 표현을 갱신한다. 여기서 매치의 세기 는 하나의 직사각형이 다른 것에 의하여 완전히 가리워 졌을 때와 같이, 매치에 대한 어떠한 증거나 그 직사각형이 존재하지 않는다는 증거가 모두 없을 때 0이 된다. 모델 포즈 리스트내의 매치되지 않은 모든 직사각형들에 대한 탐침이 수행되면, 제거되지 않은 각각의 포즈에 대하여 매치의 정도에 대한 평균이 계산된다. 여기서 가장 높은 평균을 갖는 모델 포즈가 최적의 매치로 간주된다. 이러한 계산 절차들을 나타내면 <그림 1>과 같고, 각각의 태스크에 대한 계산 복잡도를 요약하면 다음과 같다.

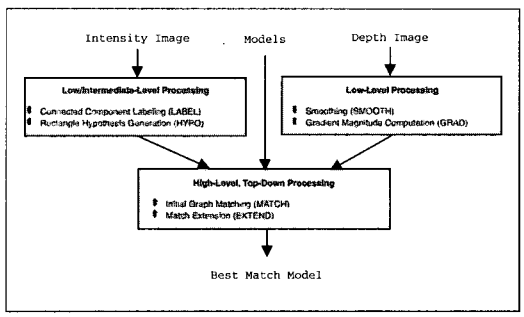

그림 1 DARPA Image Understanding 벤치마크에서 사용한 객체 인식 시스템의 흐름도

먼저 $n \times n$ 크기를 갖는 밝기(intensity)와 깊이(depth) 이미지가 주어졌을 때, 연결된 영역(region)의 최대 크기를 R_s, 밝기 이미지내의 영역의 수를 r_i라 할 때, 밝기 이미지로부터 연결된 영역을 찾아내는 LABEL의 복잡도는 $O(n^2+R_s r_i)$로 표현된다. 그리고 각각의 영역으로부터 코너들을 추출하여 직사각형의 가정을 생성해 내는 HYPO의 복잡도는, R_P를 영역의 최대 둘레, C를 영역의 최대 코너 수라 할 때, $O(n^2+r_i[Rp + clogc])$

로 표현된다. 그리고 깊이 데이타로부터 정보 추출을 위한 SMOOTH와 GRAD는, window 크기를 w라 할 때, $O(n^2w^2)$로 표현된다. 또한, 주어진 모델의 수를 m, 모델에서 최대 직사각형의 수를 r_m, 모델에서 최대 링크의 수를 l_m 이라 할 때, 부분 매치를 수행하는 상위 레벨 태스크 MATCH의 복잡도는 $O(mr_i[R_sr_m + l_m])$가 된다. 마지막으로 탐침을 통한 EXTEND는 $O(mr_mr_i [r_i + R_s\theta + \theta n])$의 복잡도를 갖는다. 이때 θ는 Hough 변화에서 정의한 각도의 수를 의미한다.

2.2 다중처리 마이크로프로세서

명령어 수준의 병렬성(Instruction-Level Parallelism, ILP) 지원을 위한 마이크로프로세서 구조로 **슈퍼스칼라**(superscalar)와 **Very Long Instruction Word(VLIW)**가 활발히 연구되고 있으며, 현재 발표되고 있는 상용 마이크로프로세서들은 대부분 슈퍼스칼라 방식을 채택하고 있다. 슈퍼스칼라[7,8] 방식의 가장 큰 특징은 동일한 파이프라인 구조를 갖는 연산장치를 여러 개 병렬로 사용하여 여러 개의 명령어를 동시에 수행시키는 기법이다. 현재 사용되고 있는 상용 마이크로프로세서 중 *Pentium*, *PA7100*, *PowerPC603*, *Alpha*는 동시에 2개의 명령어를 수행하며(2-issue), 4개 이상의 명령어를 동시에 수행시키는 *Power2*, *UltraSparc*, *PowerPC620*, *PA8000* 등도 최근에 개발되었다. 특히, 슈퍼스칼라 방식은 선형으로 구성된 명령어 흐름에서 하드웨어가 상호의존성이 없는 명령어를 골라내어 동시에 수행시키는 방식이기 때문에, 명령어간 상호의존성 조사를 위한 하드웨어가 필요하다. 또한, 슈퍼스칼라 방식의 성능을 극대화하기 위하여 분기 예측, 레지스터 재명명, 비순차 실행 등의 기술들이 요구된다. 따라서 슈퍼스칼라 방식은 동시에 처리하는 명령어의 수가 증가할수록 하드웨어 복잡도가 기하급수적으로 증가하여 실행 싸이클 시간을 증가시킨다. 또한, 명령어 구성 비율에 따라 자원활용도의 편중이 발생하여, 일반적으로 8-issue 이상에 대해서는 회의적인 전망이 지배적이다.

슈퍼스칼라 방식이 ILP를 위하여 하드웨어에 의한 동적 스케쥴링 방식을 사용하는 반면, VLIW[18] 방식은 컴파일러에 의한 정적 스케쥴링 방식을 사용한다. 즉, VLIW 방식에서는 컴파일러가 선형 명령어 흐름에서 병렬 수행 가능한 명령어들을 찾아내고, 이 명령어들을 하나의 긴 명령어로 만든다. 이때 하드웨어는 복수개의 단위 명령어로 구성된 하나의 긴 명령어를 받아 단위 명령어들을 동시에 수행시킨다. 이러한 VLIW 방식은 슈퍼스칼라 방식에 비하여 하드웨어 복잡도가 낮은 장점이 있지만, 컴파일러 개발에 어려움이 많고 마이크로프로세서 상호간 호환성을 보장할 수 없는 단점이 있다.

따라서 이러한 ILP 기반의 마이크로프로세서 구조를 대신할 새로운 구조에 대한 연구가 활발히 진행 중에 있으며[10~15], 이중 **다중처리 마이크로프로세서(on-chip multiprocessor microprocessor)**[12~14]는 ILP의 한계를 극복함과 동시에 구현의 용이성을 제공한다는 점에서 차세대 마이크로프로세서 구조로 급부상하고 있다. 즉, 간단한 구조의 프로세서 유니트를 이용하여 한 쓰레드 내의 크지 않은 ILP를 지원하면서, 여러 개의 프로세서 유니트를 이용하여 여러 개의 쓰레드를 동시에 실행시킴으로써 **쓰레드 수준의 병렬성(Thread-Level Parallelism, TLP)**[10~15]을 추가로 제공한다.

본 논문에서는 다중처리 마이크로프로세서로 *Raptor*[19]를 사용한다. Raptor는 한국전자통신연구원에서 차세대 마이크로프로세서로 개발한 단일 칩 다중처리 마이크로프로세서로, General Processor Unit(GPU)라는 4개의 독립적인 프로세서 유니트와 Graphic Co-processor Unit(GCU)라는 하나의 그래픽 연산 유니트로 구성된다. Raptor의 주요 특징은 다음과 같다:

- 각각의 프로세서 유니트는 SPARC V9 명령어 세트 구조를 가지며, 64비트 정수 및 부동 소수점 연산을 한다.
- 4개의 독립된 프로세서 유니트가 그래픽 연산 유니트를 공유하여 그래픽 연산을 처리한다.
- 크기가 비교적 작은 내장 1차 캐쉬와 대용량의 외부 2차 캐쉬로 구성된 다중 캐쉬 구조를 갖는다.
- 2차 캐쉬 제어기를 내장하며 여러 개의 Raptor 프로세서를 이용한 시스템 구현을 지원한다.

이러한 특징을 갖는 Raptor의 블록 구성도를 <그림 2>에 나타내었다. 여기서 Inter-processor Bus Unit(IBU)은 GPU와 External cache Control Unit (ECU)을 연결하는 공유 버스이다. 그리고 Multi-processor Control Unit(MCU)는 인터럽트를 GPU로 분배하고 GPU간의 동기를 지원한다. 반면, Port Interface Unit(PIU)는 칩 내부에서 발생하는 메모리 접근 요구, 입출력 장치 접근 요구, 인터럽트 및 기타 유틸리티 신호선 사용 요구를 처리한다. 특히, 부가적인 외부 로직없이 다중처리 시스템을 구성할 수 있는 multiprocessor-ready 형태의 인터페이스를 제공한다. 그리고 4개의 GPU는 각각의 레지스터 파일과 프로그램

카운터로 그래픽 명령어를 제외한 모든 명령어를 실행하지만, IBU를 통하여 ECU를 공유한다. 마지막으로 GCP는 4개의 GPU에 의해 Single Instruction stream Multiple Data stream(SIMD) 방식으로 그래픽 명령어를 처리한다. Raptor에 대한 보다 자세한 내용은 [19]에 설명되어 있다.

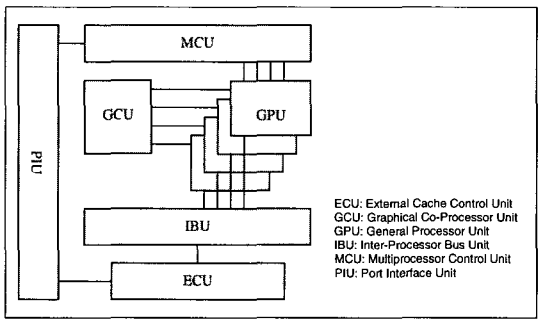

그림 2 Raptor 마이크로프로세서의 블록 구성도

3. 시뮬레이션 환경

다중처리 마이크로프로세서인 Raptor를 정성적으로 분석하기 위하여 전용 시뮬레이터인 *RapSim*을 개발하였다. RapSim은 4개의 GPU와 이들에 의해 공유되는 메모리 계층을 모델링하는 프로그램 구동 방식의 마이크로아키텍춰 시뮬레이터이다. 여기서 각각의 GPU 모델은 명령어세트 시뮬레이터인 **전처리기**(Pre- Processing Unit)와 성능 시뮬레이터인 **후처리기**(Post-Processing Unit)로 구성된다. 또한 복수개의 GPU상에서 멀티쓰레드 환경을 지원하기 위하여 *MMOS* (Multithreaded Mini-OS)라고 하는 프로그래밍 환경을 개발하였다. RapSim과 MMOS의 전체적인 관계를 <그림 3>에 나타내었다.

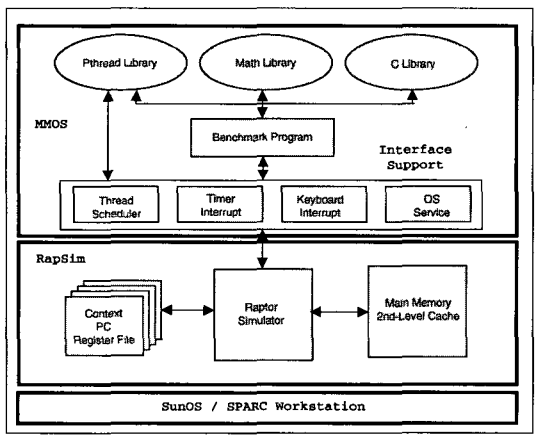

그림 3 시뮬레이션 환경

3.1 RapSim

RapSim은 컴파일러에 의해서 생성된 이진 실행 파일을 직접 입력으로 받는 프로그램 구동 방식으로 개발되었으며, 싸이클 단위로 마이크로프로세서 내부 상태를 관리하는 싸이클 기반형 시뮬레이터이기도 하다. RapSim은 <그림 4>에 나타낸 것 처럼 크게 전처리기와 후처리기로 구성되어 있으며, 전처리기에서 생성한 실시간 트레이스 정보를 사용하여 후처리기가 파이프라인 모델을 시뮬레이션 한다.

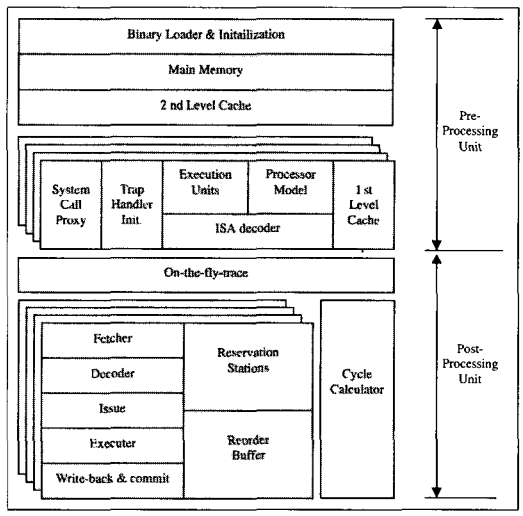

그림 4 RapSim 구조

먼저 프로그램 구동방식의 시뮬레이터인 전처리기는 명령어를 수행하는 실행기(Execution Unit)와 레지스터 파일을 비롯해서 프로세서 내부 상태를 모델링한 프로세서 모델, 캐쉬 및 메모리 모델, 그리고 시스템 콜 및 트랩 처리를 위한 부분으로 구성되어 있다. 특히, 이진 실행파일을 입력으로 받기 위해서 ELF 파일 해독기를 내장하고 있으며, 해독된 ELF 파일은 코드 부분과 데이타 부분으로 분할되어 메모리 모델에 적재된다.

프로세서 모델은 메모리 모델을 접근하여 명령어를 패치하고, 패치된 명령어는 명령어 디코더와 실행기에 의해서 처리된다. 실행기는 실행 결과를 프로세서 모델에 반영하여 레지스터 파일 및 기타 프로세서 내부 제어 레지스터 및 상태 레지스터를 갱신한다. 실행기 및

프로세서 모델에 의해서 발생되는 메모리 접근 요구는 캐쉬 모델을 통하여 이루어 지며, 이때 내부 캐쉬 및 외부 캐쉬의 히트 여부가 관리되어 진다.

전처리기는 실제 입력된 이진 파일을 수행하는 부분으로 매 명령어 처리 마다 실시간 트레이스 정보를 생성하여 후처리기로 전달한다. 실시간 트레이스는 명령어 종류, 프로그램 카운터, 오퍼랜드 형태, 캐쉬 히트 여부, 실행 처리 시간, 분기 발생 여부 등 파이프라인 제어에 필요한 정보를 모두 포함하고 있다.

후처리기는 전처리기에 의해서 생성된 트레이스를 사용하여 실제 파이프라인의 효율을 싸이클 단위로 계산하는 부분이다. 비순차 실행 및 분기 예측 기능을 갖는 2-way 슈퍼스칼라 5단 파이프라인을 모델링한 것으로, Re-Order Buffer(ROB)와 Reservation Station (RS)을 사용한다.

먼저 후처리기는 전처리기에서 발생한 실시간 트레이스를 관리하면서 순서대로 명령어를 패치하여 해당 유니트의 RS로 이슈한다. 이슈된 명령어는 명령어간 종속성 및 하드웨어 가용도에 따라서 비순차적으로 처리되며, 처리된 명령어는 ROB를 거쳐서 레지스터 파일을 갱신한 후 제거된다. 마지막으로 후처리기는 싸이클 단위로 파이프라인 모델을 제어하며, 동시에 성능 평가를 위하여 총 수행 싸이클 시간 및 실행 명령어 수와 같은 성능 인자를 관리한다. 본 논문에서 사용한 구체적인 시뮬레이션 변수의 기본값을 <표 1>에 나타내었다.

표 1 시뮬레이션 변수

항목	기본값
GPU 수	1, 2, 4
GPU 이슈 크기	2
1차 캐쉬 크기	16 Kbyte I-cache, 16 Kbyte D-cache / 32 bytes line
2차 캐쉬 크기	4 Mbyte / 32 bytes line
쓰기 갱신	1차 캐쉬에서 2차 캐쉬 : Write through 2차 캐쉬에서 메인 메모리 : Write back
1차 캐쉬 접근 시간	1 싸이클
2차 캐쉬 접근 시간	4 싸이클
메인메모리 접근 시간	10 싸이클
명령어 수행	Integer ALU = 1 싸이클 Integer Multiply = 4 ~ 34 싸이클 Integer Division = 36 (single), 68(double) 싸이클 Load/Store = 1 싸이클 Control Transfer = 1 싸이클 FP Addition/Subtraction = 1 싸이클 FP Multiply = 4 싸이클 FP Division = 12(single), 22(double) 싸이클

3.2 MMOS

MMOS는 4개의 GPU를 효과적으로 사용하기 위하여 RapSim에 멀티쓰레드 프로그래밍 환경을 제공한다. 즉, MMOS는 Pthread[20] 라이브러리, C 라이브러리, RapSim 인터페이스를 제공한다.

여기서 C 라이브러리는 여러 개의 쓰레드가 공유 C 라이브러리를 동기문제 없이 액세스할 수 있게 해주고, Pthread 라이브러리는 여러 개 쓰레드간의 동기 및 스케쥴링 기능을 제공해 준다. 그리고 RapSim 인터페이스는 MMOS를 RapSim에 연결하여 4개의 쓰레드로 4개의 프로세서를 시뮬레이션할 수 있게 하고, 쓰레드를 스케쥴하여 RapSim의 프로세서 모델로 할당한다.

4. 시뮬레이션 결과

본 논문에서는 1개의 GPU상에서 수행한 순차 실행(Sequential), 2개의 GPU상에서 2개의 쓰레드를 이용한 실행(2-Threads), 4개의 GPU상에서 4개의 쓰레드를 이용한 실행(4-Threads)에 대한 시뮬레이션을 수행하였다. 그리고 다중처리 마이크로프로세서의 성능지수로는 **수행 싸이클 수**, **Instruction Per Cycle(IPC)**, **쓰레드 오버헤드**를 설정하였다.

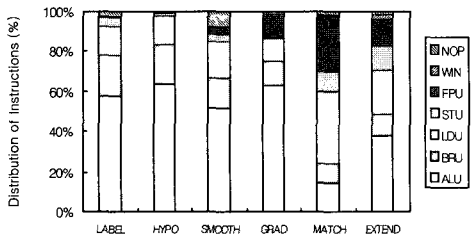

그림 5 순차 실행시 각 태스크별 명령어 분포 비교

먼저 객체 인식 응용을 구성하는 각 태스크의 계산 특성을 살펴보기 위해 순차 실행시 각 태스크에서의 명령어 분포를 나타내면 <그림 5>와 같다. 여기서 ALU은 연산 명령어, BRU는 분기 명령어, LDU는 로드 명령어, WIN은 윈도우 레지스터 처리 명령어, STU는 스토어 명령어, FPU는 부동소수점 연산 명령어를 나타낸다. 즉, 정수 데이타로 표현되는 밝기(intensity) 이미지를 처리하는 하위/중간 레벨 태스크(LABEL, HYPO)와 달리, 부동소수점 데이타로 표현되는 깊이(depth) 이미지를 처리하는 하위 레벨 태스크(SMOOTH, GRAD)는 FPU 명령어를 포함한다. 그리고, 상위 레벨 태스크(MATCH, EXTEND)는 ALU 명령어가 상대적으로 적

은 대신, graph traversal에 따른 LDU/STU 명령어와 깊이 데이타 상에서의 탐침 동작에 따른 FPU 명령어가 많은 특징을 갖는다.

다음은 쓰레드 수를 증가시킬 때 수행 싸이클 수와 IPC의 변화를 살펴보자. <그림 6>에서 알 수 있듯이, 쓰레드 수를 증가시키면 수행 싸이클 수는 선형적으로 감소하고 반대로 IPC는 선형적으로 증가한다. 즉, 객체 인식 응용은 작은 오버헤드로 쓰레드 수준의 병렬성을 적절히 활용하여 선형적인 성능 향상을 기대할 수 있다. 여기서 병렬 쓰레드 이용시 싸이클 수는 parent process를 수행하는 GPU0의 싸이클 수를 의미한다. 그리고, 객체 인식 응용의 각 태스크별 IPC를 나타내면 <그림 7>과 같은데, 각각의 태스크가 상이한 특성에도 불구하고 쓰레드 수준의 병렬성을 활용하여 유사한 IPC 값을 가짐을 알 수 있다.

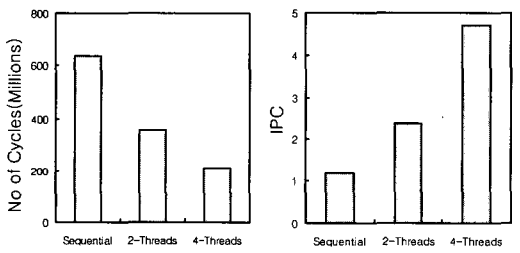

그림 6 순차 및 쓰레드를 이용한 실행시 수행 싸이클 수와 IPC 비교

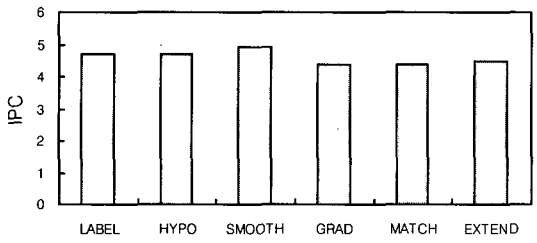

그림 7 4개 쓰레드를 이용한 실행시 각 태스크별 IPC 비교

쓰레드 수준의 병렬성을 활용하는데 따른 오버헤드를 좀 더 자세히 분석하기 위하여 다음과 같은 오버헤드 함수를 정의한다. 즉,

$$\text{Overhead}(1, 2) = (w_2 - w_1)/w_1 \times 100\%$$

여기서 w_i는 실행환경 i 에서의 작업부하라고 하며, 각 GPU에서 실행한 수행 싸이클의 합으로 정의된다. 이러한 오버헤드 함수를 요약하면 <표 2>와 같다. 즉, 쓰레드 수를 증가 시킬수록 쓰레드 동기 등의 오버헤드에 따른 오버헤드 함수값이 증가하지만, 그 값이 크지않음을 알 수 있다.

표 2 쓰레드 오버헤드

구분	값
Overhead(Sequential, 2-threads)	11.4 %
Overhead(2-threads, 4-threads)	18.0%

그리고 여러 개의 쓰레드를 이용할 경우 GPU간 부하 균형을 확인하기 위하여 4개 쓰레드를 이용한 실행시 수행 싸이클 수를 <그림 8>에 나타내었다. GPU0가 순차실행 부분 때문에 싸이클 수가 약간 많지만, 대체로 GPU간 부하 균형이 이루어짐을 알 수 있다

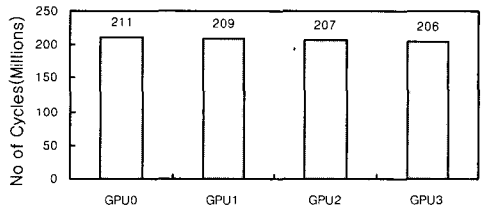

그림 8 4개 쓰레드를 이용한 실행시 GPU간 수행 싸이클 수의 분포

표 3 캐쉬 히트율

구분	히트율
GPU0 1차 캐쉬	99.99%(명령어), 98.35%(데이타)
GPU1 1차 캐쉬	99.99%(명령어), 98.59%(데이타)
GPU2 1차 캐쉬	99.99%(명령어), 98.56%(데이타)
GPU3 1차 캐쉬	99.99%(명령어), 98.48%(데이타)
외부 2차 캐쉬	99.94%

마지막으로 4개 쓰레드를 이용한 실행시 내장 1차 캐쉬와 외부 2차 캐쉬의 히트율을 요약하면 <표 3>과 같다. 즉, 2차 캐쉬 뿐만 아니라 1차 캐쉬의 히트율도 매우 높음을 알 수 있다. 특히, 4개 쓰레드를 이용한 실행시 1차 캐쉬의 데이타 히트율 평균 98.49%는 순차 실

행시 1차 캐쉬의 데이타 히트율 97.95% 보다 높음을 알 수 있다. 이는 4개의 GPU를 이용하면 1개의 GPU를 이용한 순차 실행에 비하여 capacity miss는 감소하고 coherent miss는 증가하지만, capacity miss의 감소양이 coherent miss의 증가양보다 크기 때문이다.

5. 결 론

다중처리 마이크로프로세서는 현존하는 슈퍼스칼라 구조가 갖는 명령어 수준 병렬성의 한계를 극복할 수 있으며, 수십억개의 트랜지스터가 집적된 차세대 반도체에 적합한 구조로 최근 급부상하고 있다. 즉, 하나의 마이크로프로세서 내에 간단한 프로세서 유니트를 여러 개 장착하여, 명령어 및 쓰레드 수준의 병렬성을 모두 제공할 수 있는 구조를 제공한다.

본 논문에서는 이러한 다중처리 마이크로프로세서를 이용하여 객체 인식 응용을 수행할 때 어떠한 성능 특성을 나타내는지 분석하였다. 성능 분석을 위하여 먼저 마이크로프로세서 시뮬레이터와 프로그래밍 환경을 개발하였다. 이를 기반으로 프로그램 구동 시뮬레이션을 수행하여 수행 싸이클 수, IPC, 쓰레드 오버헤드를 분석하였다. 시뮬레이션 수행 결과, GPU 수가 증가함에 따라 IPC도 거의 선형적으로 증가함을 확인하였다. 즉, 4개의 GPU를 사용할 경우 4.7 이상의 IPC를 얻을 수 있었다. 반면 GPU 수를 증가시킬수록 쓰레드 동기 등의 오버헤드가 증가함을 확인하였다.

끝으로 본 논문에서는 하나의 다중처리 마이크로프로세서에 객체 인식 작업부하를 부과한 시뮬레이션 결과를 나타내었는데, 여러 개의 다중처리 마이크로프로세서가 대규모 병렬머신으로 구현된 경우의 성능도 흥미로운 연구분야가 될 것으로 기대된다.

참 고 문 헌

[1] A. Choudhary and S. Ranka, Parallel Processing for Computer Vision and Image Understanding, *IEEE Computer*, Vol. 25, No. 2, pp. 7-11, 1992.

[2] B. Wah, et al., Report on Workshop on High Performance Computing and Communications for Grand Challenge Applications: Computer Vision, Speech and Natural Language Processing, and Artificial Intelligence, *IEEE Tr. on Knowledge and Data Engineering*, Vol. 5, No. 1, pp. 138-154, 1993.

[3] V. Kumar, *Parallel Architectures and Algorithms for Image Understanding*, Academic Press, 1991.

[4] C. Weems, Architectural Requirements of Image Understanding with respect to Parallel Processing, *IEEE Proceedings*, Vol. 79, No. 4, pp. 537-547, 1991.

[5] C. Weems, et al., Parallel Processing in the DARPA Strategic Computing Vision Programs, *IEEE Expert*, Vol. 6, No. 5, pp. 23-38, 1991.

[6] C. Wang, P. Bhat, and V. Prasanna, High Performance Computing for Vision, *IEEE Proceedings*, Vol. 84, No. 7, pp. 931-946, 1996.

[7] D. Culler, J. Singh, and A. Gupta, *Parallel Computer Architecture A Hardware/Software Approach*, Morgan Kaufmann, Pub., 1998.

[8] M. Slater, The Microprocessor Today, *IEEE Micro*, Vol. 16, No. 6, pp. 32-45, 1996.

[9] D.Wall, Limits of Instruction Level Parallelism, *WRL Research Report*, Digital Western Research Laboratory, 1993.

[10] J. Wilson, Challenges and Trends in Processor Design, *IEEE Computer*, Vol. 31, No. 1, pp. 39-50, 1998.

[11] S. Egger, et al., Simultaneous Mutithreading: A Platform for Next-Generation Processor, *IEEE Micro*, Vol. 17, No. 5, pp. 12-19, 1997.

[12] B. Nayfe, L. Hammond, and K. Olukotun, Evaluation of Design Alternatives for Multiprocessor Microprocessor, *Proc. of Intl Symp. on Computer Architecture*, pp. 66-77, 1996.

[13] L. Hammond, et al., A Single-Chip Multiprocessor, *IEEE Computer*, Vol. 30, No. 9, pp. 79-85, 1997.

[14] S. Amarashinhe, et al., Multiprocessors From a Software Perspective, *IEEE Micro*, Vol. 16, No. 3, pp. 52-61, 1996.

[15] A. Agarwal, et al., Performance Tradeoff in Multithreading Processors, *IEEE Tr. on Parallel and Distributed Systems*, Vol. 3, No. 5, pp. 525-539, 1992.

[16] J. Singh, W. Weber, and A. Gupta, SPLASH: Stanford Parallel Applications for Shared Memory, *Computer Architecture News*, Vol. 20, No. 1, pp. 5-44, 1992.

[17] C. Weems, et al., The DARPA Image Understanding Benchmark for Parallel Computers, *J. of Parallel and Distributed Computing*, Vol. 11, No. 1, pp. 1-24, 1991.

[18] J. Fisher, Very Long Instruction Word Architecture and the ELI-512, *Proc. of Intl Symp. on Computer Architecture*, pp. 140-150, 1983.

[19] 박경 외 3인, 다중처리 마이크로프로세서 설계, *대한전자공학회 추계학술대회 논문집*, pp. 717-720, 1996.

[20] IEEE, *POSIX P1003.4a: Threads Extension for Portable Operating Systems*, 1994.

정 용 화

1984년 한양대학교 전자통신공학과 졸업 (공학사). 1986년 한양대학교 전자통신공학과 졸업 (공학석사). 1997년 University of Southern California 컴퓨터공학과 졸업 (Ph.D). 1986년 ~ 현재 한국전자통신연구원 책임연구원. 관심분야는 컴퓨터구조, 병렬알고리즘, 성능분석

박 경

1991년 전북대학교 전자공학과 졸업 (공학사). 1993년 전북대학교 전자공학과 졸업 (공학석사). 1993년 ~ 현재 한국전자통신연구원 선임연구원. 관심분야는 컴퓨터구조, 마이크로프로세서 구조, 멀티미디어 서버

최 성 훈

1986년 경북대학교 전자공학과 졸업 (공학사). 1988년 경북대학교 전자공학과 졸업 (공학석사). 1988년 ~ 현재 한국전자통신연구원 선임연구원. 관심분야는 컴퓨터구조, 마이크로프로세서 구조, 멀티미디어 서버

한 우 종

1981년 고려대학교 전자공학과 졸업 (공학사). 1984년 고려대학교 전자공학과 졸업 (공학석사). 1995년 고려대학교 전자공학과 졸업 (공학박사). 1984년 ~ 현재 한국전자통신연구원 책임연구원. 관심분야는 컴퓨터구조, 마이크로프로세서 구조, 멀티미디어 서버