GPU를 이용한 DNA 컴퓨팅 기반 패턴 분류기의 효율적 구현

논 문
58-7-25

Efficient Implementing of DNA Computing-inspired Pattern Classifier Using GPU

최 선 욱* · 이 종 호†
(Sun-Wook Choi · Chong-Ho Lee)

Abstract - DNA computing-inspired pattern classification based on the hypernetwork model is a novel approach to pattern classification problems. The hypernetwork model has been shown to be a powerful tool for multi-class data analysis. However, the ordinary hypernetwork model has limitations, such as operating sequentially only. In this paper, we propose a efficient implementing method of DNA computing-inspired pattern classifier using GPU. We show simulation results of multi-class pattern classification from hand-written digit data, DNA microarray data and 8 category scene data for performance evaluation. and we also compare of operation time of the proposed DNA computing-inspired pattern classifier on each operating environments such as CPU and GPU. Experiment results show competitive diagnosis results over other conventional machine learning algorithms. We could confirm the proposed DNA computing-inspired pattern classifier, designed on GPU using CUDA platform, which is suitable for multi-class data classification. And its operating speed is fast enough to comply point-of-care diagnostic purpose and real-time scene categorization and hand-written digit data classification.

Key Words : DNA Computing, Hypernetwork, Pattern Classification, GPU

1. 서 론

디지털 컴퓨터와 인간의 뇌가 근본적으로 다른 점 중의 하나는 메모리의 구조이다. 현재의 디지털 컴퓨터에서 사용하는 메모리가 엄격히 제어되는 주소 및 인덱스를 통한 순차적 접근에 의하여 동작하는데 반해, 인간의 뇌의 메모리 구조는 다양한 개념들의 상호작용에 의한 연상 능력에 의하여 동작한다. 향후 디지털 메모리 기술이 더욱 발전하여 인간의 일생의 기억에 해당하는 매우 많은 양의 정보라 하더라도 메모리에 저장하게 될 수 있을지는 모르지만, 인간의 뇌에서 처리되는 것과 같이 필요한 정보를 매우 빠른 시간에 인출 하여 사용하는 능력을 갖는 것은 힘들 것이다.

인간의 뇌는 정보를 저장 하는 메모리 기능을 할 뿐 아니라 학습 과정을 통해 기존의 정보와의 융합에 의한 고도의 압축 기술을 구사하고 있으며 기억을 회상 할 때에도 고도의 병렬적 탐색에 기반 한 연상 작용을 이용한다. 이러한 연상 기억 정보처리는 언어를 처리하거나 시각 패턴 등을 인식하기 위한 인공지능 기술에 필수적인 능력이다. 그러나 현재의 실리콘 기반의 디지털 컴퓨터는 구조적 문제로 인해 이를 효과적으로 처리하기에는 한계가 있다 [1].

최근 이러한 기존의 컴퓨터 모델에 대한 새로운 대안으로 양자 컴퓨팅(quantum computing)이나 분자 컴퓨팅(molecular computing)과 같은 비 전통적인 컴퓨터 모델이 연구 되고 있다. 이들은 연산의 기본 단위가 되는 물질의 물리적 특성을 살려 초 병렬(massive parallel)적인 정보처리를 수행한다는 점에서 기존의 실리콘 기반의 컴퓨터 기술과 대조적이다. 특히 가장 관심 받고 있는 분자 컴퓨터 모델 중의 하나인 DNA 컴퓨팅에서는 DNA 분자들이 용액 상에서 3차원 화학반응을 하고 또한 분자들의 자기 조립(self-assembly), 분자 인식(molecular recognition) 및 자기 복제(self-replication) 하는 능력에 기반 한 대규모 상호작용을 통하여 실리콘 기반 디지털 컴퓨터로는 모방하기 힘든 초 병렬적 정보처리가 가능함을 보였다 [2].

이러한 DNA를 이용한 컴퓨팅 모델이 소개 된 후 다양한 분야의 문제들을 해결하기 위해 DNA 컴퓨팅 모델이 적용되어 왔고[3,4,5], 최근에는 패턴인식을 위한 새로운 접근 방법으로써 레퍼런스 데이터(reference data)로부터 패턴 분류기를 구성하기 위해 DNA 컴퓨팅의 기본적인 연산이라고 할 수 있는 혼성화(hybridization)와 복제(duplication)를 사용하는 DNA 컴퓨팅 기반의 분자 진화 학습 모델에 대한 연구들이 있었다 [6].

그러나 시험관 내(*in vitro*)에서의 DNA 컴퓨팅 구현에는 많은 어려움이 뒤따른다. 예를 들면 온도와 분자 농도, 염도 등의 실험 제약 조건들은 시험관 내부에서 일어나는 반응의 정확한 시뮬레이션을 위해 엄격하게 조절 되어야만 한다 [10]. 이러한 어려움들을 극복하여 DNA 컴퓨팅 모델을 활용하기 위해 확률 라이브러리 모델(probabilistic library model)과 하이퍼네트워크 모델(hypernetwork model)과 같은

* 정 회 원 : 인하대 공대 정보통신공학과 박사과정
† 교신저자, 시니어회원 : 인하대 공대 정보통신공학과 교수
E-mail : chlee@inha.ac.kr
접수일자 : 2009년 1월 9일
최종완료 : 2009년 4월 15일

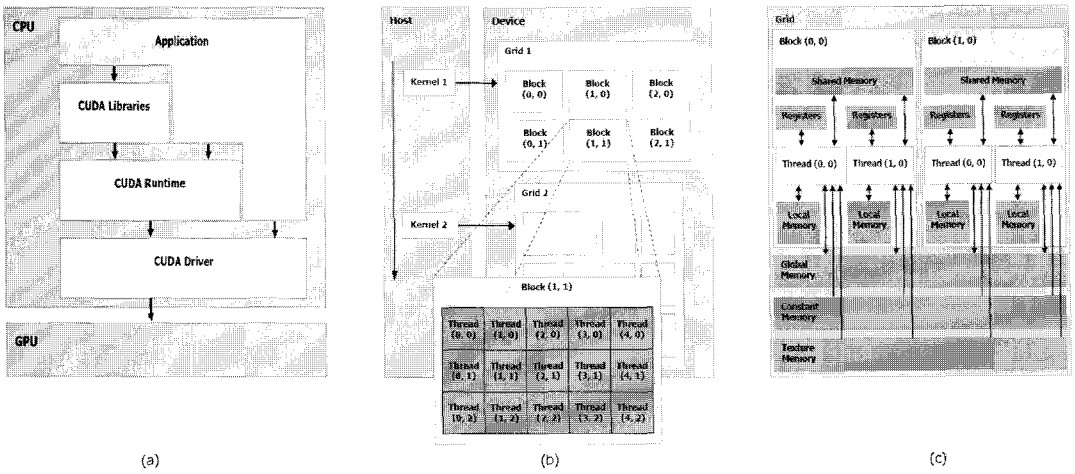


그림 1 (a) CUDA 소프트웨어 스택 (b) CUDA의 블록과 스레드 구조 (c) CUDA 메모리 모델 [19]
Fig. 1 (a) CUDA software stack (b) Block & thread structure of CUDA (c) Memory model of the CUDA

실리콘 기반 디지털 컴퓨터상에서 시뮬레이션 된 방법들이 제시 되었다 [8,9].

DNA 컴퓨팅 기반 패턴 분류기는 앞의 실리콘 컴퓨터상에서 동작하는 하이퍼네트워크 모델에 기반 한 것으로서 패턴 인식의 다양한 영역에 적용 되었고, SVM, ANN, k-NN과 같은 기존의 기계학습 알고리즘들과 비교하여 경쟁력 있는 성능을 보여주었다 [10,11]. 그러나 기존의 하이퍼네트워크 모델을 이용한 방법들은 DNA 컴퓨팅을 이용한 접근 방법이 초 병렬 연산이라는 강력한 특징을 갖고 있음에도 불구하고 순차적 머신(sequential machine) 상에서만 시뮬레이션 되어 왔기 때문에 필연적으로 본래의 장점을 잘 활용하지 못한다는 한계를 갖고 있다.

본 논문에서는 이러한 한계를 극복하기 위해 초 병렬 연산이 이루어 질 수 있도록, 최근 병렬 연산을 위한 대안으로 관심 받고 있는 GPU(Graphics Processing Unit)를 이용한 DNA 컴퓨팅 기반 패턴 분류기를 제안 하고, 멀티 클래스의 데이터 분류에 적용하여 그 성능을 분석한다.

2. 관련 연구

최근 GPU의 성능이 급격히 발전하고, NVIDIA의 CUDA와 ATI의 Stream과 같은 플랫폼이 발표 되면서 GPU를 이용한 개발이 매우 용이해졌다. 이러한 환경의 변화로 인해 GPU 기반의 병렬 프로그래밍에 대한 관심이 증가하고 있으며, 이를 연구에 응용하기 위한 논의도 활발하게 진행되고 있다. 특히 반복 되는 대규모 연산이 필요한 동영상 인코딩이나 3D 렌더링 작업 등에서는 이미 GPU가 큰 영향력을 가지고 있고, 관련 소프트웨어들도 GPU를 지원하고 있는 상황이다. 또한 대규모의 병렬 연산이 필요한 패턴인식 분야에서의 GPU 적용에 관한 연구들 또한 증가하고 있는 추세이다.

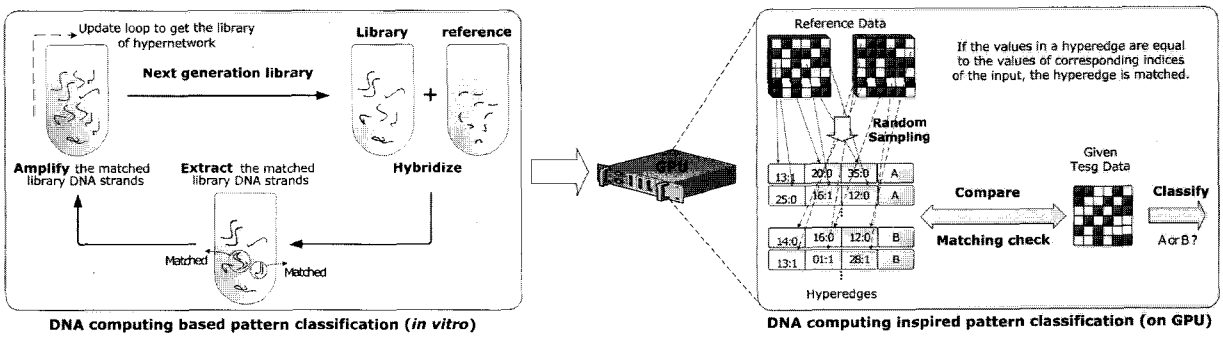
Gustavo Poli 등은 CUDA 기반의 네오코그니트론 신경망을 얼굴 인식에 적용 하였고 [12], Andreas Brandstetter 등은 GPU 기반의 RBFN(radial basis function networks)을 개발하여 근사 문제에 적용 하였다 [13]. Oh 등은 GPU 기반의 신경망을 사용하여 이미지나 영상내의 텍스트를 검출하는데 적용 하였고, Rolfes 등은 GPU 기반의 컨벌루션 신경망을 이용하여 문자 인식에 적용 하였다 [14]. Kris Woodbeck 등은 GPU 기반의 Visual Cortex를 개발하여 영상 속에서 고속으로 특징(feature)들을 추출 하고, 이를 이용하여 영상을 분류하는데 적용 하였다 [15]. Luo 등과 Prabhu는 GPU 기반의 비교사 학습(unsupervised learning)의 대표적인 방법인 자기구성 맵(self-organizing map)을 구현하여 패턴 분류에 적용 하였다 [16,17].

위의 연구들에서는 공통적으로 기존의 패턴인식을 위한 분류기들이 순차적 머신 상에서 동작하면서 필연적으로 맞닥뜨린 즉, 구조적 한계로 인한 수행 성능 저하를 극복하기 위해 GPU를 적용하였고, 상당한 수행 성능의 향상을 보였다.

3. 범용적 연산을 위한 GPGPU

최신 GPU의 연산처리 속도는 대응 되는 최신 CPU의 성능을 수 배, 경우에 따라서는 수십 배 능가 한다 [18]. 예를 들어 NVIDIA의 GPU 코어인 G80(GeForce 8800 GTX)의 경우 7억 개의 트랜지스터 집적도와 350 GFLOPS에 근접하는 초고속 연산이 가능한 것에 비하여, 최신 CPU 코어인 Intel Core2 Quad는 약 96 GFLOPS 가량의 연산을 처리 할 수 있다 [19]. 또한 최근 GPU의 발전 추세에 따른 하나는 GPU 내부 파이프라인의 기능을 사용자가 직접 프로그래밍 하여 사용할 수 있도록 허용한다는 것이다. 그러나 GPU를 연산에 이용하기 위한 기존의 방법들은 그래픽 렌더링 파이프라인의 구조에 대한 이해 등을 요구하는 등 다른 분야에서 활동하는 사용자가 접근하기에는 어려움이 있었다.

최근 NVIDIA나 ATI가 경쟁적으로 자사의 GPU를 사용하게 하기 위한 플랫폼(NVIDIA의 CUDA와 ATI의 stream 등)들을 발표함으로써 GPU의 대용량 병렬 연산을 이용한 범용 어플리케이션의 개발이 보다 가속화 되고 있다. 또한



(a) (b)

그림 2 (a) 시험관 내에서의 DNA 컴퓨팅 기반 패턴 분류 (b) GPU를 이용한 DNA 컴퓨팅 기반 패턴 분류

Fig. 2 (a) DNA computing based pattern classification in vitro (b) DNA computing based pattern classification on GPU

최근 크로노스 그룹(Khronos Group)의 주도로 GPU를 범용적으로 사용하기 위한 오픈 스탠다드인 OpenCL이 발표 되어 GPGPU를 이용한 연산에 대한 관심은 그 어느 때보다도 큰 상황이다. 범용 목적으로의 GPU의 응용, 즉 GPGPU를 위해 컴퓨터 그래픽스 분야 외에 최근 가장 각광을 받고 있는 응용 분야는 영상처리와 컴퓨터 비전, 패턴 인식과 같은 분야이다. 이러한 분야에 적용 되는 대부분의 알고리즘들은 동일한 명령의 많은 양의 데이터로의 동시 실행, 즉 SIMD(single instruction multiple data) 방식의 접근을 필요로 하며 이는 GPU의 본래 이용 목적이었던 3차원 그래픽 알고리즘과의 공통적인 특징이기도 하다. 또한, GPU가 PC 뿐만 아니라 핸드폰, PDA와 같은 모바일 기기용으로 개발되고 있고 이러한 것들이 실제 적용되고 있는 기술 동향을 볼 때, 향후 모바일 기기에서의 영상 처리나 부가적으로도 응용 될 수 있는 패턴 인식과 같은 많은 연산이 필요한 작업에 대한 처리도 GPU가 담당 할 수 있게 될 것으로 기대된다[18]. 본 논문에서는 NVIDIA에서 제공하는 CUDA (Compute Unified Device Architecture) [19] 기반의 GPU를 사용한다. CUDA는 기존의 복잡한 텍스쳐(Texture), 쉐이딩(shading) 작업을 통한 GPU의 이용 대신, C언어 형식으로 GPU 프로그래밍이 가능하도록 하는 환경을 지원한다. 이러한 장점으로 인해 최근 많은 연구들이 NVIDIA의 CUDA기반의 GPU를 사용하여 실험한 결과들을 발표하고 있다.

CUDA는 그림 1 (a)와 같이 CUDA 라이브러리와 드라이버를 통하여 GPU에 접근 가능하다. GPU 상에서 실제 연산 되는 부분은 커널(kernel) 형태로 작성 되어 CPU에서 호출 된다. 각 커널은 그림 1 (b)와 같이 블록(Block) 들로 이루어진 하나의 그리드(grid) 이며, 블록들은 여러 개의 스레드(thread)로 구성 된다. 따라서 병렬 연산의 기본 구조는 이 블록들과 스레드들의 조합을 통해 구성 된다. 블록은 내부의 스레드들 사이에서 그림 1 (c)와 같이 공유 메모리(shared memory)와 레지스터(register)를 공유 할 수 있게 되어 있는 구조 이고, 따라서 블록 내의 스레드들은 공유 된 메모리 자원을 활용하여 연산 하게 된다. 이러한 구조적 특징으로 인해 CUDA 기반의 GPU를 이용한 연산의 성능을 최대한 끌어올리기 위해선 디바이스 내에서 활용 가능한 글로벌 메모리(global memory)와 상수 메모리(constant memory) 등을 스레드 내의 공유 메모리 등과 얼마나 잘 결합하여 활용하느냐에 달려 있다고 할 수 있다. 또한 이러한 GPU가 제공하는 병렬 연산 능력을 최대한 활용하기 위해서는 병렬 연산에 적합한 새로운 알고리즘을 개발하거나, 기존의 알고리즘을 병렬 연산 구조에 적합하도록 변환 하여 GPU 상에서 구현 할 수 있도록 해야 한다.

4. GPU를 이용한 DNA 컴퓨팅 기반 패턴 분류기

4.1 하이퍼네트워크(hypernetwork)

DNA 컴퓨팅 기반 패턴 분류기(DNA computing-inspired pattern classifier, DNACPC)는 하이퍼네트워크를 이용하여 설명 할 수 있다. 하이퍼네트워크는 그림 2 (a)와 같이 시험관 내에서 동작하는 DNA 컴퓨팅을 실리콘 컴퓨터상에서 시뮬레이션 할 수 있는 모델로써, 수많은 DNA 분자 조각들로 메모리 조각을 표현하고 이러한 메모리 조각들이 상보결합하는 과정을 통해 기억을 인출하는 메커니즘을 실리콘 컴퓨터상에서 모델링 할 수 있다[6, 20].

하이퍼네트워크는 하이퍼그래프(hypergraph)와 유사한 구조를 갖고 있다. 하이퍼그래프는 에지(edge)가 널(null) 값을 가지지 않는 정점(vertices)들로 연결 된 비 방향성 그래프 $G=(V,E)$ 이다. 여기서 $V=\{v_1,v_2,...,v_l\}$는 정점들의 집합, $E=\{e_1,e_2,...,e_m\}$는 하이퍼에지(hyperedge)들의 집합을 나타낸다. 각각의 하이퍼에지는 $e_i=\{v_{i_1},v_{i_2},...,v_{i_n}\}$와 같이 n개의 정점들의 결합으로 구성 된다. 일반적으로 그래프의 에지는 2개까지의 정점을 연결 할 수 있지만, 하이퍼에지는 2개 이상의 정점들 간의 연결이 가능하다는 것이 특징이다. 하이퍼네트워크는 그림 3에서와 같이 하이퍼그래프의 각 하이퍼에지에 가중치를 할당하여 일반화 한 형태로, 가중치에 의해 각 정점들이 얼마나 강하게 연결되어 있는지를 표현 할 수 있다. 그림 3에서 하이퍼에지의 가중치는 선의 두께로 표현하였다. 하이퍼네트워크는 하이퍼그래프에 가중치 W가 추가된 $H=(V,E,W)$로 나타낼 수 있으며, 여기서 V, E는 하이퍼그래프에서 설명 된 것과 동일하고, $W=\{w_1,w_2,...,w_m\}$는 하이퍼에지의 가중치를 나타낸다. 하이퍼네트워크를 이용한 패턴 분류기에서 학습 과정은 주어진 레퍼런스 데이터 집합을 하이퍼네트워크의 라이브러리로 저장하는 과정으로 볼 수 있으며, 분류 과정은 주어진 테스트 데이터에 대해 하이퍼네트워크로부터 해당 데이터를 연상 해 내는 과정이라고 볼 수 있다.

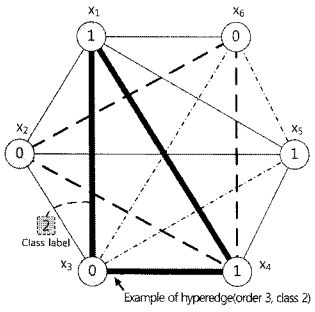


그림 3 하이퍼네트워크의 예
Fig. 3 Example of the hypernetwork

하이퍼네트워크의 라이브러리 L은 하이퍼에지들로 구성되는데, $l_i=\{l_{i_1},l_{i_2},...,l_{i_n},y_i\}$는 n개의 특징(feature)들과 클래스 레이블(label)을 포함하고 있는 하이퍼에지이다. 여기서 하이퍼에지의 j 는 문제 공간상의 특징들의 인덱스 값이다. 실제 시스템에서는 그림 2 (b)에서와 같이, 라이브러리 L을 구성하고 있는 하이퍼에지들을 표현하기 위해 각 레퍼런스 데이터에 대해 랜덤 하게 생성 된 인덱스 값들로 이루어진 배열을 이용해 하이퍼에지를 구성하는 특징들을 선택하는 마스킹 기법을 사용한다. 하이퍼에지들은 각 레퍼런스 데이터들에 대해 랜덤 하게 생성 된 마스크 배열의 수를 곱한 만큼 만들어진다. 이 때문에 하이퍼에지를 랜덤 마스크라 표현하기도 한다. 라이브러리 L에 포함 된 전체 하이퍼에지의 수는 레퍼런스 데이터의 수(N_r)와 랜덤 마스크의 수(N_m)의 곱($N_r\times N_m$)만큼 존재한다고 생각할 수 있다.

하이퍼에지가 포함하는 특징들의 수 n은 차수(order)라고 하고, 일반적으로 하이퍼네트워크에서는 모든 하이퍼에지가 동일한 차수 k를 갖는 k-uniform 하이퍼에지를 사용한다. 하이퍼에지의 모든 특징들의 값들과 이와 상응하는 데이터 패턴의 모든 요소들이 동일 할 때 그 하이퍼에지는 매치(matched) 되었다고 말할 수 있다.

$$\begin{aligned}\boldsymbol{x_i}&=\{x_{i_1}=1,x_{i_2}=0,x_{i_3}=0,x_{i_4}=1,x_{i_5}=1,x_{i_6}=0\}\\ l_j&=\{l_{j_1}=1,l_{j_3}=0,l_{j_4}=1,l_{j_y}=2\}\end{aligned}\quad(1)$$

예를 들어 차원이 6인 입력 데이터 패턴이 식(1)의 $\boldsymbol{x_i}$와 같고, 대응되는 하이퍼에지가 식 (1)의 l_j와 같다고 할 때, 이 경우 하이퍼에지는 클래스 레이블이 $l_{j_y}=2$ 이고, 3개의 특징을 포함하고 있다고 할 수 있다. 이는 해당 하이퍼에지의 차수가 3임을 의미한다. 또한 $l_{j_1}=x_{i_1}$, $l_{j_3}=x_{i_3}$ 그리고 $l_{j_4}=x_{i_4}$와 같이 하이퍼에지를 구성하고 있는 모든 특징들이 주어진 입력 데이터의 상응하는 구성 요소와 동일하기 때문에 하이퍼에지 l_j는 입력 데이터 패턴 x_i와 매치 되었다고 할 수 있다. 이 때, 매치 된 하이퍼에지의 클래스 레이블이 2 이기 때문에 입력 된 데이터 패턴의 클래스가 2로 예측 될 확률이 증가하게 된다고 할 수 있다.

하이퍼네트워크가 적절하게 구성 되면, 이를 패턴 분류기로 사용하는 것이 가능하다. 입력 데이터 패턴 $\boldsymbol{x}$에 대해, 하이퍼네트워크 H 는 예측 된 클래스 레이블 y^*를 출력하고, 이는 입력 된 테스트 데이터 패턴에 대한 각각의 클래스의 조건부 확률로서 설명 할 수 있다. 테스트 데이터 패턴 $\boldsymbol{x}$가 주어진다면, 출력 클래스 y^*는 각각의 클래스의 조건부 확률의 계산에 의해 예측되어 진다. 그리고 이때 선택 되어지는 클래스는 아래 식 (2) 에서와 같이 가장 높은 조건부 확률 값을 갖는다.

$$\begin{aligned}y^* &= \underset{Y\in\{0,1,...,n\}}{\arg max}\ P(Y|\boldsymbol{x})\\ &= \underset{Y\in\{0,1,...,n\}}{\arg max}\ \frac{P(Y,\boldsymbol{x})}{P(\boldsymbol{x})}\end{aligned}\quad(2)$$

여기서 $P(Y,\boldsymbol{x})=P(Y|\boldsymbol{x})P(\boldsymbol{x})$ 이고, $Y\in\{0,1,...,n\}$는 후보 클래스를 나타낸다. 하이퍼네트워크 모델을 이용하여 가장 높은 조건부 확률 값을 갖는 클래스를 예측하기 위한 과정은 다음과 같이 요약 할 수 있다.

1. 라이브러리 L을 하이퍼에지들을 이용하여 구성
   - 라이브러리 L은 입력 패턴 $\boldsymbol{x}$와 출력 클래스 Y의 결합 확률 $P(\boldsymbol{x},Y)$를 경험적 확률 분포로 나타냄
2. 테스트 할 입력 패턴 $\boldsymbol{x}$가 주어짐
3. 입력 패턴 $\boldsymbol{x}$와 매치 되는 모든 하이퍼에지를 라이브러리로부터 추출
4. 추출 된 하이퍼에지들의 클래스를 M에 각각 카운트
   - 클래스 레이블이 $Y=0$ 인 경우 M^0로 카운트
   - 클래스 레이블이 $Y=1$ 인 경우 M^1로 카운트
   . . .
   - 클래스 레이블이 $Y=n$ 인 경우 M^n로 카운트
5. 가장 많은 수가 카운트 된 클래스를 입력 패턴 $\boldsymbol{x}$의 클래스로 분류

단계 3 에서는 입력 패턴 $\boldsymbol{x}$와 매치 되는 모든 하이퍼에지를 추출하여, 추출 되는 수를 M에 카운트 한다. 이를 통해 입력 패턴과 매치 되는 하이퍼에지를 발견할 확률을 구할 수 있다.

$$\frac{c(\boldsymbol{x})}{|L|}=\frac{|M|}{|L|}\approx P(\boldsymbol{x})\quad(3)$$

단계 4 에서는 입력 패턴 $\boldsymbol{x}$가 주어졌을 때 각 클래스의 발생 정도를 나타내는 $c(Y|x)$를 구하는 과정이다. 이 값을 이용해 입력 데이터 패턴이 주어졌을 때 예측 될 클래스 레이블들에 대한 조건부 확률인 사후 확률을 아래와 같이 근사할 수 있다.

$$\frac{c(Y|\boldsymbol{x})}{|M|}=\frac{|M^Y|}{|M|}\approx P(Y|\boldsymbol{x})\quad(4)$$

단계 5 에서는 가장 많은 수가 카운트 된 클래스를 입력 데이터 패턴의 클래스로 예측하게 되는데, 이는 식 (2)의 사후확률이 최대가 되게 하는 클래스 레이블을 찾는 것과 같다고 생각할 수 있다. 더 자세한 이론적 배경은 다음에서 참조 할 수 있다 [6,21].

4.2 GPU를 이용한 DNA 컴퓨팅 기반 패턴 분류기 구현

앞 절에서 설명 된 하이퍼네트워크를 기반으로 하는 DNA 컴퓨팅 기반 패턴 분류기는 분자들의 3차원 공간상에서의 생화학적 작용에 기반 한 DNA 컴퓨팅의 본래의 특성에 기인하여 초 병렬 연산 구조를 갖고 있고, 주로 저장, 매

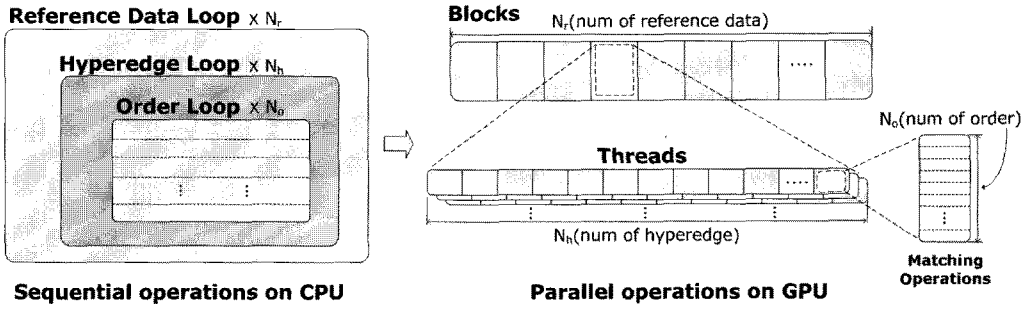


그림 4 (a) 기존 순차적 머신 상에서의 DNA 컴퓨팅 기반 패턴 분류기의 연산 구조
(b) 제안 된 GPU를 이용한 DNA 컴퓨팅 기반 패턴 분류기의 연산 구조

Fig. 4 (a) Operation structure of the DNAC based pattern classification on ordinary sequential machine
(b) Proposed operation structure of the DNAC based pattern classification using GPU

칭, 선택과 복사와 같은 단순한 병렬 메모리 연산들로 이루어져 있다. 이들은 본래 이상적으로는 모든 조건이 갖추어져있다고 가정 했을 경우, 모든 연산이 시험관 내에서 한 번에 이루어지는 구조이다. 그러나 기존의 하이퍼네트워크를 이용한 패턴 분류기들은 순차 머신 상에서 구현 되었고, 수행 속도 보다는 기존의 기계학습 방법들과 비교하여 경쟁력 있는 성능을 갖고 있음을 보이는 것과 패턴 내 특징들 간의 고차 상관(higher-order correlation) 관계를 효과적으로 모델링 할 수 있는 방법임을 보이는 것에 주안점을 두어왔다 [6,8,10,11]. 기존의 순차 머신 상에서 구현 된 하이퍼네트워크 모델은 그림 4 (a)에서와 같이 입력 패턴과 라이브러리 내의 하이퍼에지들을 비교하고, 매칭 되었을 경우 해당 클래스에 대해 카운트 하는 반복적인 연산 과정을 3중의 루프를 통하여 순차적으로 수행한다. 이 작업의 시간 복잡도는 $\Theta(N_r \cdot N_h \cdot N_o)$ 로써 레퍼런스 데이터의 수(N_r)가 증가 되거나, 비교 되는 하이퍼에지의 수(N_h) 그리고 하이퍼에지의 차수(N_o)가 증가 할수록 많은 연산 시간을 필요로 하게 된다. 이는 DNA 컴퓨팅 기반 패턴 분류기의 실제 문제에 활용하는 것을 저해하는 원인이 된다. 이러한 한계를 극복하기 위해서 DNA 컴퓨팅 기반 패턴 분류기가 갖고 있는 본래의 특성인 초 병렬 연산을 효과적으로 수행 할 수 있는 방법이 필요하고, 본 논문에서는 앞서 소개한 GPU가 제공하는 병렬 연산 능력을 이용하여 DNA 컴퓨팅 기반 패턴 분류기를 보다 효율적으로 구현 가능함을 보인다.

4.2.1 CUDA기반의 GPU를 이용한 구현

앞서 그림 1 (b)에서 본 것과 같이 CUDA 기반의 GPU는 일련의 스레드들로 구성 된 블록과 블록들의 집합인 그리드를 구성하는 것을 통해 병렬 연산이 가능하도록 지원해준다. 일반적으로 CUDA는 NVIDIA의 G80 계열의 GPU에서 한 개의 블록 당 최대 512개의 스레드를 사용 할 수 있도록 허용하기 때문에 이에 유의하여 커널을 구성해야할 필요가 있다. 또한 메모리 액세스 타임을 줄이기 위해, 생성 된 각 스레드에서는 글로벌 메모리에 저장 되어 있는 데이터들에 직접 접근 하여 사용하는 것 보다, 블록 내 스레드 간 공유가 가능하고 빠른 액세스 타임을 보장하는 공유 메모리로 불러와 사용할 수 있도록 하는 것이 효과적이다. 이처럼 CUDA 기반의 GPU 프로그래밍에서는 한정 된 자원을 효율적으로 활용 하는 것이 최적의 성능을 내기 위해 매우 중요하다 할 수 있다.

GPU를 이용한 DNA 컴퓨팅 기반 패턴 분류기 구현을 위해 그림 4 (a)의 레퍼런스 데이터 루프는 CUDA에서의 커널 내의 블록으로 대치하여 구현 할 수 있다. N_r번 반복 되는 레퍼런스 데이터 루프를 구현하기 위해 N_r개의 블록을 사용해 병렬적으로 동작 할 수 있도록 한다. 입력 데이터 패턴이 주어졌을 때, 하이퍼에지들과 입력 데이터 패턴 대해 랜덤 마스크 데이터로부터 얻은 인덱스에 상응하는 각각의 값들을 차수만큼 비교 연산하고, 이를 통해 매칭여부를 결정하는 2중의 루프(Hyperedge Loop & Order Loop)는 블록 내의 N_h개의 스레드와 각 스레드 내에서 매칭 여부를 판단을 위해 매칭 테이블로부터 매칭 결과를 얻어 오는 N_o번의 연산을 사용하여 구현 한다. 이러한 일련의 과정들은 GPU 상에서 병렬적으로 동작하기 때문에, 그 결과 고속의 처리 속도를 얻을 수고 또한 이를 통해 본래 DNA 컴퓨팅이 지향하는 초 병렬 연산 특성에 근사하는 분류기를 구현 할 수 있게 된다. 제안하는 GPU를 이용한 DNA 컴퓨팅 기반 패턴 분류기는 최종적으로 그림 4 (b)와 같은 연산 구조를 갖는다.

4.2.2 매칭 테이블 구현과 적용

DNA 컴퓨팅 기반 패턴 분류기의 가장 기본적인 연산으로써 분류기의 전체 연산 과정에서 가장 빈번하게 발생하는 연산이라고 할 수 있는 것은 각 하이퍼에지를 구성하고 있는 특징들과 이에 상응하는 주어진 테스트 데이터 특징들의 비교를 통해 매칭 여부를 결정 하는 연산이다. 이는 앞서 설명한 GPU를 이용한 구현에서 각각의 스레드 내에서 이루어지는 N_o번의 연산에 해당한다.

데이터 집합과 하이퍼에지를 구성하는 특징 값의 구성 레벨 수에 따라 각 특징들의 매칭 여부를 판단하는 방법에 차이가 존재하게 된다. 이로 인해 데이터 집합 마다 매칭 여부를 판단하는 조건식이 서로 상이해 일관성 있는 구현이 힘들고, 수행 성능에서도 차이가 발생하게 된다. 예를 들어 이진화 된 2-레벨 데이터를 사용하는 경우, 상응 하는 특징

들의 값이 '0'과 '0' 이거나 '1'과 '1' 이라면 해당 특징들은 매칭 되었다고 할 수 있다. 4 단계의 값을 갖도록 이산화 된 4-레벨 데이터를 사용하는 경우, 기본적으로 상응 하는 특징들의 값이 '1' 과 1', '2' 와 '2', '3' 과 '3', '4' 와 '4' 일 때 매칭 된 것으로 생각 할 수 있으나, 이산화의 특성 상 선정된 문턱 값 인근의 값들이 서로 인접하고 있음에도 불구하고 서로 다른 값으로 이산화 되어, 매치가 되지 않는 것으로 판단되는 경우도 발생 할 수 있다. 따라서 더 높은 분류 성능을 얻기 위해서는 앞의 인접한 데이터의 이산화 문제를 고려한 매칭 연산이 이루어져야 한다.

T_i \ R_i	1	2	3	4
1	1	1	0	0
2	1	1	1	0
3	0	1	1	1
4	0	0	1	1

그림 5 4-레벨 매칭 테이블의 예
Fig. 5 4-level matching table

매칭 여부를 판단하는 비교기는 테이블 형태로 구성 하여 사용하는 것이 효과적이다. 이러한 테이블을 매칭 테이블이라 하고, 하이퍼에지와 입력 데이터 패턴 내에서 서로 상응하는 특징 값(각각 그림 5의 R_i와 T_i에 해당)들을 두 개의 입력으로 갖는다. 이 값들을 매칭 테이블에 접근 하기위한 인덱스 값으로 사용하여 해당 위치로부터 리턴 되는 값을 통해 매칭 여부를 결정 하게 된다. 그림 5에서와 같이 테이블 값 설정을 통해 인접 값들이 매칭 될 수 있도록 하는 것 또한 용이하다. 이 매칭 테이블은 모든 블록 내의 스레드들에서 동일하게 사용 되는 것으로 커널을 호출하기 전 상수 메모리에 저장 하여 각 스레드에서 빠르게 접근 할 수 있도록 한다. 그림 5는 4-레벨 특징 값을 사용 할 경우에 이용될 수 있는 매칭 테이블의 예이고, 아래의 코드는 4-레벨 매칭 테이블을 CUDA를 이용하여 구현한 예이다.

```
__constant__ __device__ int table[4*4] = {1, 1, 0, 0,
                                          1, 1, 1, 0,
                                          0, 1, 1, 1,
                                          0, 0, 1, 1};
```

이와 같은 매칭 테이블에 대한 접근은 그림 4 (b)에서 볼 수 있듯이 한 스레드 내에서 하이퍼에지의 차수만큼 반복되게 되는데, 이는 루프 언롤링(loop unrolling) 기법을 통하여 구현하는 것이 효율적이다.

4.2.3 카운터와 최종 분류기 구현

하이퍼네트워크의 각 하이퍼에지들과 주어진 테스트 데이터가 매치 되는지를 확인 하는 과정은 전체 레퍼런스 데이터 수와 랜덤 마스크를 이용한 하이퍼에지들의 수의 곱($N_r \times N_h$)과 같은 수의 스레드를 통해 병렬적으로 이루어진다. 여기서 얻어진 결과들은 각 클래스별로 카운트 된다. 카운트 과정에서는 빠른 계산을 위해 매칭 결과들을 순차적으로 더하지 않고, 블록 내의 각 스레드들에서 최적화 된 트리 기반의 병렬 감소(parallel reduction) 기법을 사용해 고속으로 카운트 결과를 얻을 수 있도록 한다. CUDA를 이용한 병렬 감소 기법 구현에 있어 고려해야 할 점은 스레드 내에서 접근 가능한 공유 메모리는 32비트 단위로 16개의 뱅크가 만들어져 있다는 것이다. 32비트 보다 작은 단위의 데이터를 사용 할 경우 블록 내의 인접 스레드들에서 동일한 뱅크에 접근 하게 되기 때문에, 뱅크 컨플릭트(bank conflict) 문제가 발생하게 된다. 이 때 해당 스레드들은 순차적으로 연산되게 되는데, 이로 인해 발생한 지연은 성능 감소의 주요 원인이 된다. 따라서 병렬 감소 기법을 사용함에 있어 배열의 기본 단위로 32비트 변수를 사용하였다. 또한 공유 메모리가 16개의 뱅크로 나뉘어 접근할 수 있도록 되어있기 때문에 공유 메모리에 접근 할 때에는 32개의 스레드가 묶인 warp가 두 개로 나뉘어 half-warp 단위로 접근하게 된다. 이로 인해 블록 내 스레드들에서의 메모리 접근 방식에 따라 서로 다른 스레드에서 동일 뱅크에 접근하게 되어 뱅크 컨플릭트 문제가 발생할 수 있다. 따라서 그림 6에서와 같이 순차 어드레싱 방식의 병렬 감소 기법을 사용하여 뱅크 컨플릭트로 인한 지연을 최소화 할 수 있도록 하였다 [22].

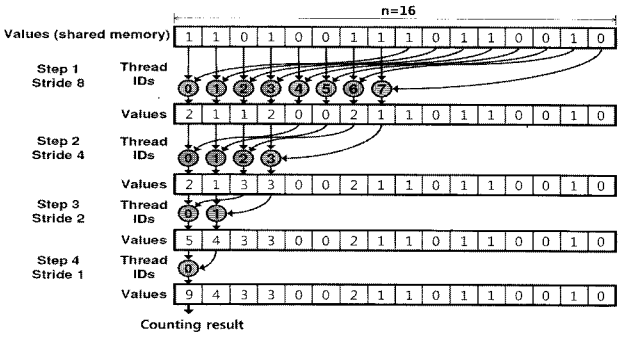


그림 6 순차어드레싱 방식의 병렬 감소기법을 이용한 카운트의 예
Fig. 6 Example of Counting using sequential addressing based parallel reduction method

카운팅에 사용 되는 배열의 크기는 각각의 블록 내의 스레드 수(N_h)와 동일한 크기로 보통 수백 개 정도의 크기를 갖는다. 이와 같은 연산이 매 블록에서 반복되기 때문에 빠른 카운팅 속도가 필요하고, 순차 어드레싱 방식을 사용 하여 카운터를 구현 할 경우 매우 효과적이라고 할 수 있다.

이렇게 각 클래스 별로 카운트 된 결과를 이용하여, 다시 앞의 병렬 감소 기법을 통해 식 3과 같이 가장 많은 수가 카운트 된 클래스를 분류 결과로 출력되도록 최종 분류기를 구현 한다.

5. 실험 및 결과

본 논문에서는 구성 데이터의 특징 수와 하이퍼네트워크를 구성하는 레퍼런스 데이터의 수 등이 서로 다른 다양한 데이터 집합에 대한 성능을 평가하기 위하여 필기체 숫자 인식 데이터와 DNA 마이크로어레이로부터 얻어진 백혈병 데이터, 그리고 장면(scene) 분류의 성능을 평가하기 위해

표 1 실험에 사용 된 데이터 집합 비교
Table 1 Comparison of datasets used in the experiment

데이터 집합	클래스 수	특징 수	샘플 수	
			레퍼런스 데이터 수	테스트 데이터 수
필기체 숫자 인식 데이터	10	64	3823	1797
DNA마이크로어레이 데이터	6	90	163	85
장면(scene) 분류 데이터	8	170	800	1888

표 2 실험에 사용 된 GPU와 CPU 사양 비교
Table 2 Comparison of GPU & CPU spec

GPU	멀티프로세서 수	코어 클록	메모리 크기
8600 GT	4	540 MHz	256 MB
9800 GTX	16	675 MHz	512 MB
CPU	프로세서 수	코어 클록	메모리 크기
Q6600	4	2.4 GHz	4 GB

(※ 사용 된 CPU는 4개의 프로세서를 가지고 있는 쿼드 코어 프로세서지만 비교를 위해 단일 코어로만 사용)

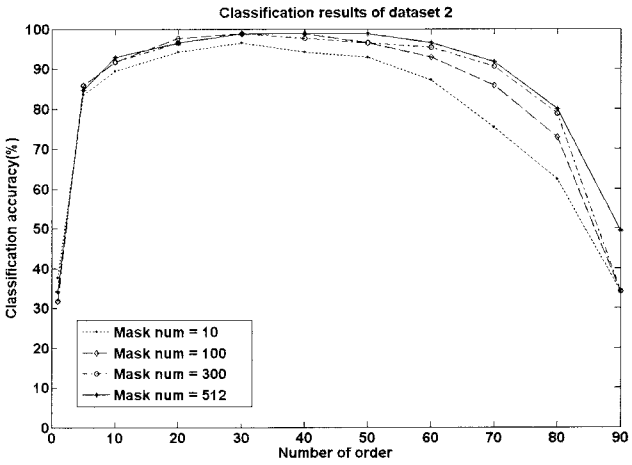


그림 8 DNA 마이크로어레이 데이터의 실험결과
Fig. 8 Experiment results of DNA microarray data

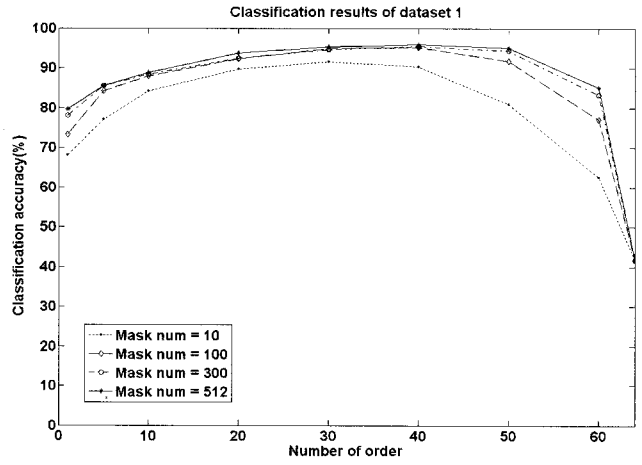


그림 7 필기체 숫자인식 데이터의 실험결과
Fig. 7 Experiment results of handwritten digit data

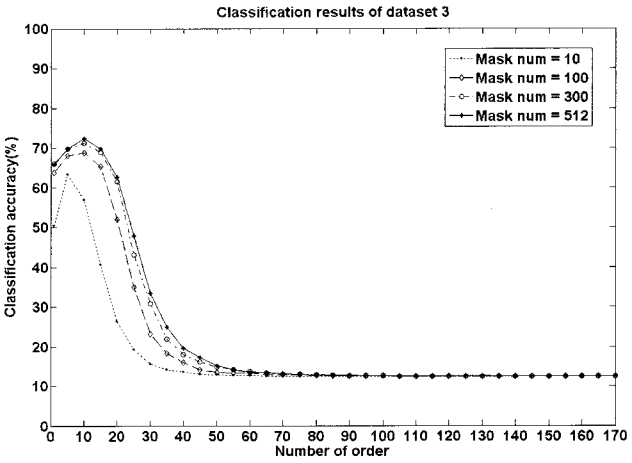


그림 9 장면 분류 데이터의 실험결과
Fig. 9 Experiment results of scene classification

만들어진 데이터를 실험에 사용하였다. 표 1은 실험에 사용된 데이터 집합들의 구성을 보여준다.

일반적으로 각 데이터들에 대해 최적의 분류 성능을 보이는 하이퍼에지의 수와 차수는 고정 되어 있지 않다. 각 레퍼런스 데이터에 대한 하이퍼에지의 수는 많을수록 좋으나 너무 많은 양을 사용한다면 연산 량이 증가하는 문제가 있다. 또한 데이터들마다의 특성에 의해 서로 상관관계를 갖으면서 유의한 특성을 보이는 특징들의 수가 다르기 때문에 이 값들을 실험을 통해 결정 할 필요가 있다. 그림 7~9는 각 데이터 집합들에 대해 최적의 하이퍼에지 수와 차수를 결정하기 위해 MATLAB으로 구현 한 DNA 컴퓨팅 기반 패턴 분류기를 이용하여 실험 한 결과로써, 각 레퍼런스 데이터에 대한 하이퍼에지의 수와 차수 각각의 값들을 변경해 가며 테스트 데이터를 분류기에 적용하여 얻은 분류 정확도이다. Mask num은 하이퍼에지를 만들기 위해 사용한 랜덤 마스크의 수를 의미하는 것으로써, 하나의 레퍼런스 데이터에 대한 하이퍼에지의 수를 의미한다. 앞서 설명한 대로 하이퍼에지의 수가 많을수록 좋은 분류 성능을 보이나, 일정 이상이 되면 성능향상의 정도가 줄어들고 있음을 확인 할 수 있다. 또한 최고의 분류 성능을 보이는 차수가 데이터

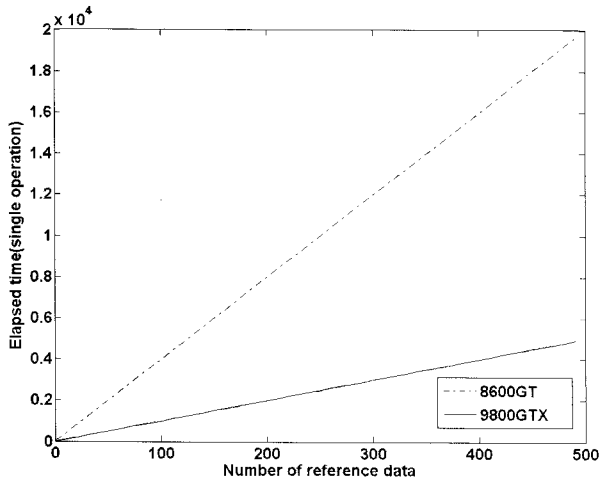


그림 10 레퍼런스 데이터 수의 변화에 따른 GPU 별 예상 수행 시간 비교

Fig. 10 Comparison of estimated time according to change in the number of reference data on each GPU type

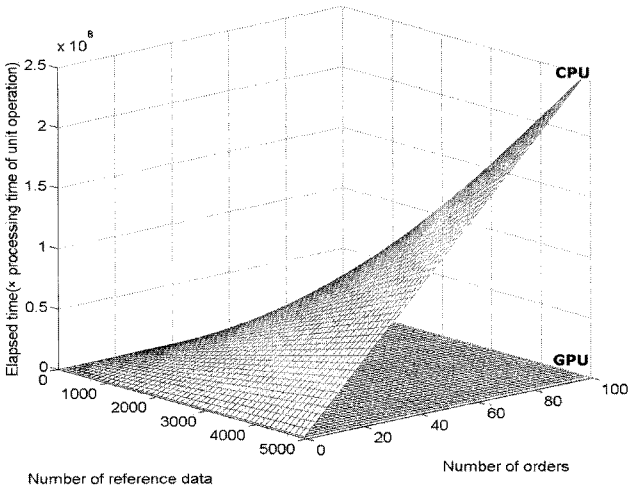


그림 11 레퍼런스 데이터 수와 차수 변화에 따른 CPU와 GPU의 예상 수행 시간 비교

Fig. 11 Comparison of estimated time between CPU and GPU according to change in the number of reference data and order

집합 마다 다름을 확인 할 수 있다. 이는 각 데이터 집합의 패턴을 구성하고 있는 특징들의 상관관계에 있어 유의미한 특성을 보이는 특징들의 수가 각각 다르기 때문이다.

앞의 실험을 통해 얻어진 각 데이터 집합들에 대한 하이퍼에지의 수와 차수 값들을 적용해, 각각의 데이터에 대해 수행환경 별 성능을 비교하기 위하여 두 가지의 서로 다른 GPU와 CPU 상에서 이루어진 실험을 통하여 수행 속도 차이를 측정 하였다. 사용 된 GPU는 각각 CUDA를 사용 가능한 NVIDIA의 GeForce 8600 GT와 GeForce 9800 GTX이고, CPU로는 Intel Core2 Quad Q6600(2.4GHz)을 사용 하였다.

표 2는 실험에 사용 된 GPU의 사양을 비교해 놓은 것이다. 수행 속도 측정에 있어, 9800 GTX는 8600 GT 보다 4배 많은 멀티프로세서(multi processor)를 포함 하고 있기 때문에 수행 속도가 4배가량 우수하게 관측 될 것이라고 예상할 수 있다. CUDA에서는 스레드들을 관리하기 위해 SIMT(single-instruction, multiple-thread)라고 하는 새로운 구조를 적용했고, 이를 위해 32개의 스레드를 warp라는 단위로 묶어 동작하도록 한다. warp는 각 프로세서 내의 8개의 멀티프로세서에서 연산되기 때문에, warp 내의 모든 스레드들이 연산되기 위해서는 최소 4 cycle의 클록이 필요하고, 실제로 동시에 연산 되는 전체 스레드의 수는 멀티프로세서의 수와 warp 내의 32개 스레드의 곱이라고 할 수 있다. 표 2에서 볼 수 있듯이 9800 GTX는 16개의 멀티프로세서를, 8600 GT는 4개의 멀티프로세서를 갖고 있기 때문에, 이론적으로는 각각 최대 512개와 128개의 스레드가 동시에 연산 된다고 할 수 있다. 그러나 실제로 블록 내의 스레드들은 한정 된 레지스터와 공유메모리를 이용하기 때문에 이들의 사용량에 따라 동시에 연산 가능한 스레드 수는 이론적인 수보다 줄어들 수 있다. 따라서 비교를 위해 연산시간을 결정짓는 다른 요인들은 모두 동일한 조건이라고 가정하고, 최대 사용 가능한 스레드를 모두 활용한다고 가정 했을 경우, 연산 속도는 각 GPU의 멀티프로세서 개수에 비례하여 차이가 발생한다고 생각 할 수 있다. 실험에 사용 된 9800 GTX와 8600 GT는 멀티프로세서 개수에 있어 4배의 차이가 나기 때문에, 실험 결과에 있어서도 약 4배의 차이가 날 것이라고 예상 할 수 있다. 그림 10은 레퍼런스 데이터 수의 변화에 따른 GPU별 예상 수행 시간 비교이다. 레퍼런스 데이터 수가 증가함에 따라 8600 GT와 9800 GTX 모델의 수행 시간의 차이가 커지게 될 것임을 예상 할 수 있다.

표 3은 제안 된 DNA 컴퓨팅 기반 패턴 분류기와 동일한 데이터를 사용한 다른 연구결과들과의 분류 정확도를 비교해 놓은 것이다. 이미 널리 알려져 사용 되고 있는 다른 기계 학습 알고리즘들과의 비교에서도 매우 경쟁력 있는 분류 정확도를 보이고 있음을 확인 할 수 있다.

5.1 데이터 1 : 필기체 숫자 인식 데이터

실험에 사용 된 데이터 1은 필기체 숫자 인식 데이터로서, UCI(University of California, Irvine) 기계학습 데이터베이스에서 제공하는 필기체 숫자 이미지 데이터를 사용 하였다 [23]. 64픽셀의 이미지로 구성 되어 있고, 0~9까지 10개의 클래스를 갖는 다중 클래스 데이터 이다. 또한 레퍼런스데이터 3823개, 테스트 데이터 1797개로 구성 되어 있다. 실험을 위해 0~16의 정수 값으로 구성 된 특징들을 4단계의 값을 갖는 특징으로 이산화 하여 사용 하였다. 최적의 차수를 결정하기 위한 소프트웨어 시뮬레이션 결과 차수 34에서 95.77%의 분류 정확도를 얻을 수 있었다.

5.2 데이터 2 : DNA 마이크로어레이 데이터

실험에 사용 된 데이터 2는 DNA 마이크로어레이 데이터이다. Yeoh 등이 공개한 데이터로서 백혈병(leukemia)의 한 종류인 ALL(Acute Lymphoblastic Leukemia)의 6개의 부

표 3 알고리즘들 간의 분류 성능 비교
Table 3 Comparison of the classification performance between algorithms

데이터 집합	DNACPC(%)	SVM(%)	ANN(%)	k-NN(%)
필기체 숫자 인식 데이터	95.77	91.40	90.5	93.50
DNA마이크로어레이 데이터	97.25	98.00	98.50	97.16
장면(scene) 분류 데이터	72.45	N/A	N/A	71.51

타입으로 구성되어 있는 데이터 이다 [24]. 일반적으로DNA 마이크로어레이로부터 얻어진 유전자 발현 데이터의 유전자 수는 수천에서 수만 개로 매우 많은 반면 샘플 수는 극히 적다는 것이 특징이다. 또한 획득된 유전자 발현 데이터에서 각 샘플의 특정 클래스와 연관이 있는 유전자의 수는 그보다 훨씬 적다. 따라서 특징 추출(feature extraction) 과정이 필수적인데, 본 논문에서 사용 된 유전자 발현 데이터에 대해 Golub 등이 [25]에서 제안한 Signal-to-noise 기법을 사용하여 각 클래스와 상관관계가 높은 90개의 유전자를 선택하여 사용 하였다. 각 유전자 발현 값들을 4개의 단계를 갖는 값으로 이산화 하여 사용하였다. 레퍼런스 데이터 163개와 테스트 데이터 84개로 구성 되어 있다. 최적의 차수를 결정하기 위한 소프트웨어 시뮬레이션 결과 차수 33에서 97.25%의 분류 정확도를 얻을 수 있었다. 이러한 DNA 마이크로어레이를 통해 얻은 데이터를 분석함에 있어 그 동안의 일반적인 기계학습 기법들은 분류에서는 어느 정도 성능을 보이나, 그것을 해석하는 것에는 어려움이 보였다. 하이퍼네트워크로 설명 되는 DNA 컴퓨팅 기반 패턴 분류기는 기존의 기계 학습들과 비교해 경쟁력 있는 분류 성능을 보일 뿐만 아니라, 어떤 유전자들의 조합이 질병에 관계되어 있는지 분석 할 수 있는 구조를 갖고 있기 때문에 의미 있는 해석을 이끌어 낼 수 있다는 장점이 있다.

5.3 데이터 3 : 장면(scene) 분류 데이터

실험에 사용 된 데이터 3은 장면 분류에 적용하기 위해 Oliva 등이 공개한 데이터로, 해변, 대지, 숲, 산, 고속도로, 거리, 시내, 고층빌딩의 8가지 카테고리로 이루어져 있는 이미지 데이터베이스를 사용 하였다 [26]. 각 이미지는 256×256 픽셀의 크기를 갖고 있고, 총 2688개의 이미지로 구성되어 있다. 분류기의 성능을 측정하기 위해 동일한 데이터를 사용한 다른 연구들과 동일하게 800개의 데이터를 레퍼런스 데이터로 사용하고 1888개의 데이터를 테스트 데이터로 사용 한다. 또한 DNA 컴퓨팅 기반 패턴 분류기에서 사용하기 위해 각 이미지로부터 MPEG-7의 에지 히스토그램 기술자를 사용한 방향 히스토그램을 이용하여 각 이미지에 대해 상/하, 좌/우, 가로, 세로, 전체 등의 영역에 대해 각각 5가지 방향의 히스토그램 빈(bin)으로 이루어진 170개의 특징들을 검출 하고 8개의 단계를 갖는 값으로 이산화 하여 사용 하였다 [27]. 최적의 차수를 결정하기 위한 소프트웨어 시뮬레이션 결과 차수 12에서 72.45%의 분류 정확도를 얻을 수 있었다. 다른 데이터 셋들에 비해 장면 데이터를 구성하고 있는 특징들이 서로 다른 영역에 대해 연관성이 높지 않은 독립 된 조합으로 구성되기 때문에 차수가 높아질수록 분류 성능이 급격하게 떨어짐을 앞의 그림 9를 통해 알 수 있다.

5.4 GPU를 이용한 실험 결과

앞의 시뮬레이션 결과들을 통해 얻은 값을 바탕으로 GPU를 이용하여 구현한 DNA 컴퓨팅 기반 패턴 분류기에 각 데이터를 적용 한 결과는 그림 12와 같다. GPU의 사양이 높아질수록 수행시간이 단축 되었으며, CPU의 경우 가장 분류 수행 시간이 길다는 것을 확인 할 수 있다. 특히 레퍼런스 데이터의 양이 많은 데이터1(필기체 숫자 인식 데이터)의 경우 다른 데이터들과 비교하여 분류 수행시간의 차이가 큼을 확인 할 수 있다.

GPU에 따라 분류 수행 성능이 차이가 나는 이유는 앞서 설명 되어진 것과 같이 멀티프로세서의 수에 따른 동시에 연산 되는 스레드 수의 차이와, 덧붙여 코어 클록, 메모리의 크기와 대역폭 등의 차이 때문이라고 할 수 있다. 이중 가장 큰 차이를 만들어내는 것은 멀티프로세서 수의 차이라고 할 수 있는데, 이는 GPU의 병렬 연산 수행 능력에 있어 가장 직접적인 척도가 되는 것이 동시에 연산 되는 스레드의 수이기 때문이다.

실험 결과 3개의 데이터 집합에 대해서, CPU와 비교하여 GeForce 8600 GT가 약 3~4배 정도 빠른 수행 시간을 보였고, GeForce 9800 GTX가 7~15배 정도 빠른 수행 시간을 보였다. 실험 결과를 보면 데이터 별로 수행 시간 단축의 정도가 다른 것을 확인 할 수 있다. 이는 각 데이터의 레퍼런스 데이터 수와 데이터를 구성 하는 패턴의 길이 그리고 차수가 다르기 때문에 발생 하는 현상이라고 생각 할 수 있다. 위와 같은 요인들의 변화가 성능에 영향을 주는 이유는 아래와 같이 생각해 볼 수 있다.

첫 번째, 글로벌 메모리로의 잦은 액세스 문제이다. 데이터 매칭 시 레퍼런스 데이터와 주어진 테스트 데이터로부터 값을 얻어와 매칭 여부를 확인하게 되는데, 이때 레퍼런스 데이터의 크기가 상수 메모리의 크기보다 크기 때문에 글로벌 메모리에 저장 하고 로드하게 된다. 레퍼런스 데이터의 수가 많을수록 메모리 액세스 타임이 긴 글로벌 메모리로의 액세스가 증가 되고, 수행 속도는 떨어지게 된다. 또한 같은 이유로 각 스레드 내에서 레퍼런스 데이터로 접근 하는 횟수라고 할 수 있는 차수가 증가 하게 될수록, 수행 속도는 저하된다.

두 번째, 테스트 데이터는 모든 그리드와 블록에서 동일하게 사용하기 때문에, 공유 메모리에 저장하고 사용하는데

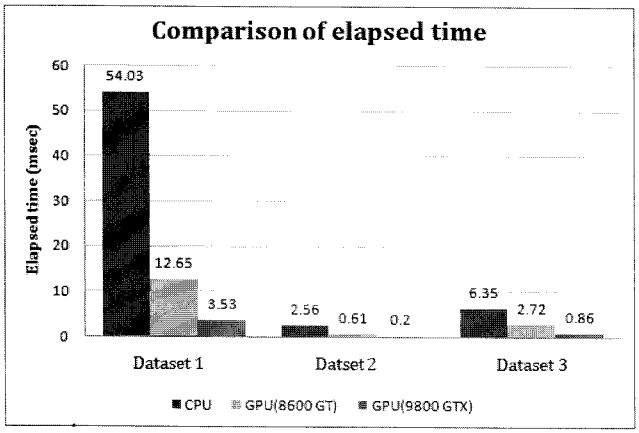


그림 12 환경 별 수행 시간 비교

Fig. 12 Comparison of elapsed time on each environment

있어서 테스트 데이터 패턴의 길이가 길수록 공유 메모리사용량이 증가하고 이로 인해 동시에 연산 가능한 스레드의 수가 감소하게 된다.

세 번째, 블록 내의 각 스레드에서 공유 메모리에 접근하며 발생 할 수 있는 뱅크 컨플릭트 문제이다. 공유 메모리 사용 시, 각 스레드들이 동일한 뱅크에 접근 하려 한다면 병렬 연산이 불가능하게 되고 순차적인 연산이 이루어지게 된다. 따라서 뱅크 컨플릭트 현상이 많이 발생 할수록 수행 속도가 저하된다. 그런데 해당 데이터 집합에 대한 하이퍼에지의 차수가 클수록 공유 메모리로 접근하는 양이 많아지게 되고, 뱅크 컨플릭트 문제가 발생할 확률이 증가 하게 된다. 이는 차수의 차이에 따른 수행 속도 저하의 원인이 된다.

이러한 문제들은 향후 GPU에 최적화 된 커널의 설계와 메모리 관리로 개선 될 수 있을 것으로 보며, 더 상위의 GPU를 사용 할 경우 더 높은 성능의 향상을 기대 할 수 있을 것이다.

6. 결 론

초 병렬적 탐색에 기반 한 연상 작용을 통해 패턴 인식을 수행 하고자 하는 새로운 접근 방법인 DNA 컴퓨팅 기반 패턴 분류기는 기존의 실리콘 기반 디지털 컴퓨터 상에서 그 구조적 한계로 인해 본래의 연산 특징을 살리기 힘들었고, 시뮬레이션 단계에만 머물러 빠른 수행 성능을 요구하는 실제 문제에 적용하기 힘들었다.

본 논문에서는 DNA 컴퓨팅을 기반으로 하는 새로운 패턴 분류기의 초 병렬 연산의 장점을 살리기 위해, 기존 CPU 기반의 순차적 연산 방식을 개선하여, CUDA 기반의 GPU를 이용한 DNA 컴퓨팅 기반 패턴 분류기의 구조를 제안하였다. 이를 멀티클래스 데이터 분류에 적용 한 결과 기존의 CPU에서의 동작에 비해 GPU에서 구현 된 경우 7~15배 수행 속도의 향상을 보였고, 이러한 수행 속도 단축의 결과로 대용량의 데이터를 실시간으로 처리하기에 충분한 성능을 얻을 수 있음을 확인 할 수 있었다.

실시간 처리 성능을 얻기 위해 CUDA 기반의 GPU를 이용하는 방법은 성능뿐만 아니라 기존의 실시간 처리를 위한 FPGA 혹은 ASIC을 통한 전용 하드웨어 기반의 접근 방법과 비교해 개발 기간을 크게 단축시킬 수 있고, 새로운 데이터가 적용 되더라도 그 구조를 빠르게 변경 가능하다는 장점을 갖고 있다. 이러한 장점들로 인해 앞으로도 병렬 연산 능력을 필요로 하는 많은 알고리즘들이 GPU 상에서 구현 될 것으로 기대 된다.

본 논문에서 사용 된 GPU의 개발사인 NVIDIA 경우, 기존의 단일 GPU 방식을 넘어 여러 개의 GPU를 병렬로 묶어 초 병렬 연산 전용의 GPU 모델들을 출시하고 있는데, 이를 활용 할 경우 더욱 향상 된 병렬 연산 능력으로 인해 매우 큰 크기의 데이터들로 구성 된 패턴일 지라도 고속으로 처리가 가능 할 것으로 기대 된다. 또한 ATI의 Stream과 NVIDIA의 CUDA 사이의 경쟁을 위한 활발한 움직임과 크로노스 그룹의 GPGPU를 위한 오픈 스탠다드인 OpenCL 등의 발표로 인해 앞으로의 GPU의 연산 능력을 이용한 GPGPU 연구의 흐름이 주목 된다.

참 고 문 헌

[1] Zhang, B.-T., "Molecular nanobiointelligence computers: computer science meets biotechnology, nonotechnology, and cognitive science (in Korean)", Communications of the Korea Information Science Society, 23(5):41-56, 2005.

[2] Sienko, T., Adamatzky, A., Rambidi, N.G., and Conrad, M., "Molecular Computing", MIT Press, 2003.

[3] Adleman, L., "Molecular computation of solutions to combinatorial problems", Science, 266: 1021-1024, 1994.

[4] Braich, R.S., Chelyapov, N., Johnson, C., Rothemund, P. W.K., and Adleman, L., "Solution of a 20-variable 3-SAT problem on a DNA computer", Science, 296: 499-502, 2002.

[5] Faulhammer, D., Cukras, A.R., Lipton, R.J., and Landweber, L.F., "Molecular computation: RNA solutions to chess problems", Proc. Natl. Acad. Sci. USA, 97(4):1385-1389, 2000.

[6] Zhang, B.T., Jang, H.Y., "A bayesian algorithm for in vitro molecular evolution of pattern classifiers", DNA computing 10. LNCS 3384, pp. 458-467, 2002.

[7] Wetmur, J., "Physical chemistry of nucleic acid hybridization. DNA Based Computers III", DIMACS Series in Discrete Mathematics and Theoretical Computer Science 48, pp. 1-23, 1999.

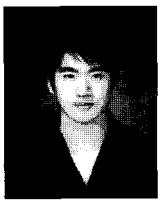
[8] Zhang, B.T., Kim, J.K., "Dna hypernetworks for information storage and retrieval", DNA computing 12. LNCS 4287, pp. 298-307, 2003.

[9] Zhang, B.T., "Solving logic problems by DNA: self-assembly process of double helix is a computational algorithm (in Korean)", Donga Science, 22(6), pp. 78-81, 2007.

[10] Ha, J., Eom, J., Kim, S., Zhang, B., "Evolutionary hypernetwork models for aptamer-based cardiovascular disease diagnosis", In: Proc. of the GECCO 2007,

ACM New York, NY, USA, pp. 2709-2716, 2007.
[11] Kim, J.K., Zhang, B.T., "Evolving hypernetworks for pattern classification", In: Proc. of IEEE CEC 2007, pp. 1856-1862, 2007.
[12] Poli, Gustavo; Saito, José Hiroki; Mari, João F.; Zorzan, Marcelo R., "Processing Neocognitron of Face Recognition on High Performance Environment Based on GPU with CUDA Architecture," Computer Architecture and High Performance Computing, 2008. SBAC-PAD '08. 20th International Symposium on , vol., no., pp.81-88, Oct. 29 2008-Nov. 1 2008.
[13] Andreas Brandstetter, Alessandro Artusi, "Radial Basis Function Networks GPU-Based Implemen tati on", IEEE TRANSACTIONS ON NEURAL NETWO RKS, VOL. 19, NO. 12, Dec., 2008.
[14] K.-S. Oh, K. Jung, "GPU implementation of neural networks," Pattern Recognition, vol. 37, no. 6, pp. 1311 - 1314, 2004.
[15] Woodbeck, K., Roth, G., Huiqiong Chen, "Visual cortex on the GPU: Biologically inspired classifier and feature descriptor for rapid recognition", Computer Vision and Pattern Recognition Workshops, Computer Vision and Pattern Recognition Workshops, 1-8, 2008.
[16] Zhongwen Luo, Hongzhi Liu, Zhengping Yang, Xincai Wu, "Self-Organizing Maps computing on Graphic Process Unit," in 13th European Symposium on Artificial Neural Netwo,rBkselgium,2005, pp. 557-562.
[17] Prabhu, R.D., "SOMGPU: An unsupervised pattern classifier on Graphical Processing Unit," Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on , vol., no., pp.1011-1018, 1-6 June 2008.
[18] 이만희, 박인규, 원석진, 조성대, "GPU를 이용한 DWT 및 JPEG2000 구현", 전자공학회논문지, 제44권 SP편 제 6호, pp. 9 ~ 15, 2007. 11.
[19] NVIDIA, "NVIDIA CUDA Programming Guide 2.0", 2008
[20] Zhang, B.T., Kim, J.K., "Dna hypernetworks for information storage and retrieval", DNA computing 12. LNCS 4287, pp. 298-307, 2003.
[21] Zhang, B.T., "Hypernetworks: A molecular evolutio nary architecture for cognitive learning and memory", Computational Intelligence Magazine, IEEE 3(3), pp. 49-63, 2008.
[22] Mark Harris, Optimizing Parallel Reduction in CUD A, 2008.
[23] UCI Machine Learning Repository, http://archive.ic s.uci.edu/ml.
[24] Yeoh, E., Ross, M., Shurtleff, S., Williams, W., Patel, D., Mahfouz, R., Behm, F., Raimondi, S., Relling, M., Patel, A., et al., "Classification, subtype discovery, and prediction of outcome in pediatric acute lymphoblastic leukemia by gene expression profiling", Cancer Cell 1(2), pp. 133-143, 2002.
[25] Golub, T., Slonim, D., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J., Coller, H., Loh, M., Downing, J., Caligiuri, M., et al., "Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring", Science 286(5439), pp. 531-537, 1999.
[26] Oliva, A., Torralba, A, "Modeling the shape of the scene: A holistic representation of the spatial envelope", International Journal of Computer Vision 42(3), pp. 145-175, 2001.
[27] Won, C.S., "Feature Extraction and Evaluation Using Edge Histogram Descriptor in MPEG-7", Advances in Multimedia Information Processing -PCM 3333, pp. 583-590, 2004.

저 자 소 개



최 선 욱 (崔 善 旭)

1983년 9월 27일생. 2007년 인하대 전자공학과 졸업. 2009년 동 대학 정보통신공학과 졸업(공석). 2009년~현재 동 대학원 정보통신공학과 박사과정.
Tel : 032-860-7396
E-mail : swchoi@inhaian.net

이 종 호 (李 鍾 浩)

1953년 4월 14일생. 1976년 서울대 전기공학과 졸업. 1978년 동 대학원 전기공학과 졸업(공석). 1986년 미국 아이오와 주립대 전기 및 컴퓨터 공학과 졸업(공박). 1986년~1989년 미국 노틀담 대학교 조교수. 1997년~1998년 인하대 직접회로 설계센터 소장. 2004년~2005년 브라운 대학교 두뇌 및 신경회로망 연구소 방문교수. 1989년~현재 인하대학교 정보통신공학과 교수. 2000년~현재 수퍼지능기술연구소 소장
Tel : 032-860-7396
E-mail : chlee@inha.ac.kr