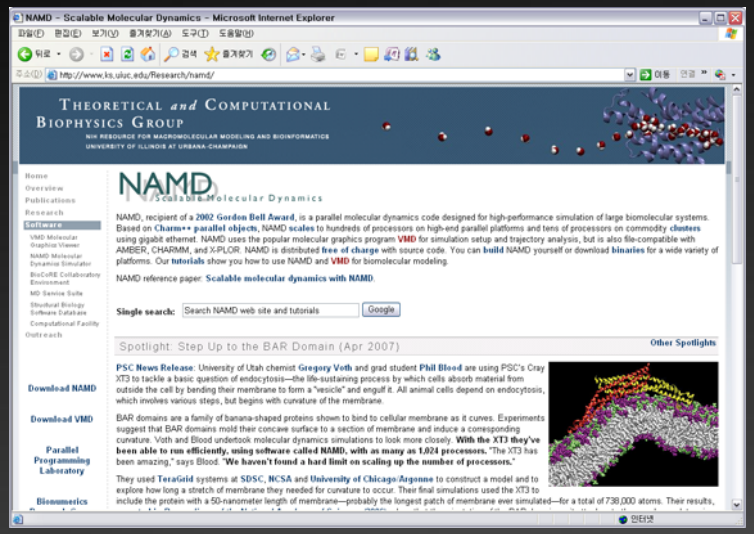
Neural Networks using CUDA and OpenMP

정기철
(kcjung@ssu.ac.kr/ http://hci.ssu.ac.kr)
숭실대학교 IT대학 미디어학부
(http://www.ssu.ac.kr/ http://media.ssu.ac.kr)

VMD/NAMD Molecular Dynamics

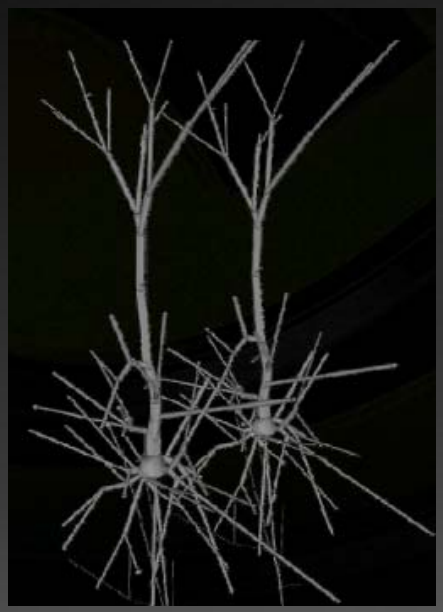
- 일리노이주립대
- 가시분자동력학(VMD)/나노분자동력학(NAMD)
- 240X 속도향상

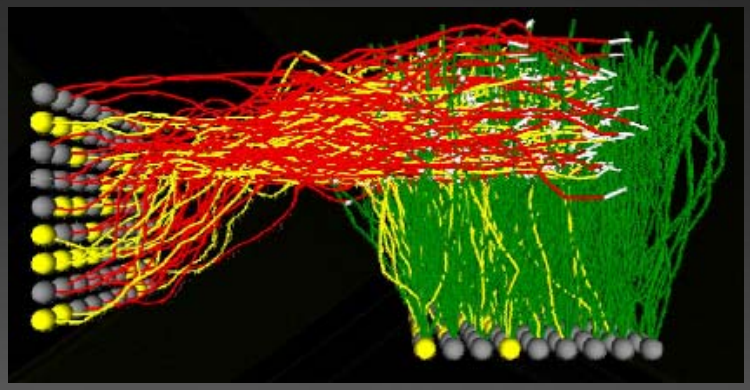


- http://www.ks.uiuc.edu/Research/vmd/projects/ece498/lecture/

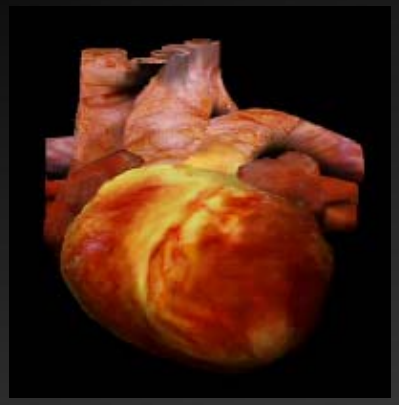
Evolved machines

- 신경회로 시뮬레이션
- 130X 속도향상

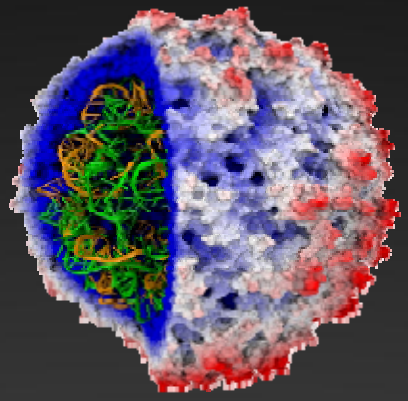




ETC.



MRI:40~170X



Virus:110X

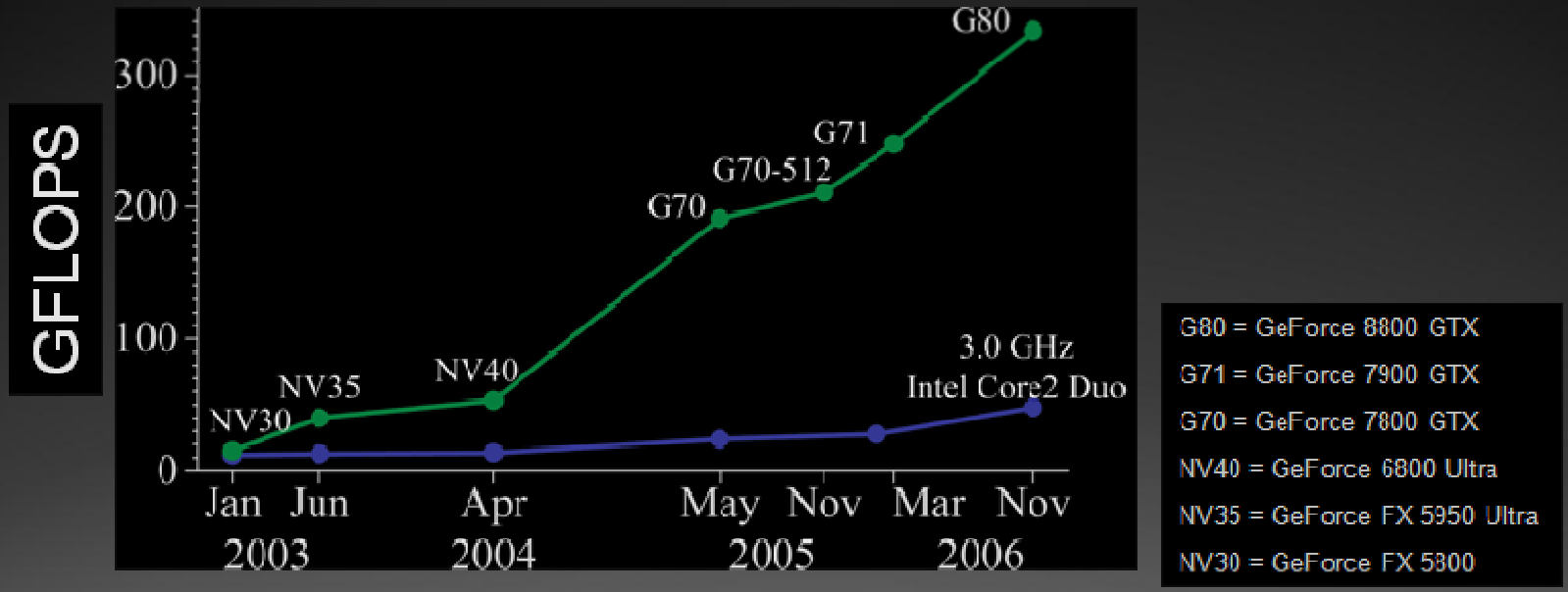
EM:45X

GPU란?

- Graphics Processing Unit
- 1999/08 NVIDIA에서 처음 발표
- CPU의 그래픽 처리 작업을 돕기 위해 만듦
- 3D그래픽 가속 칩을 개량

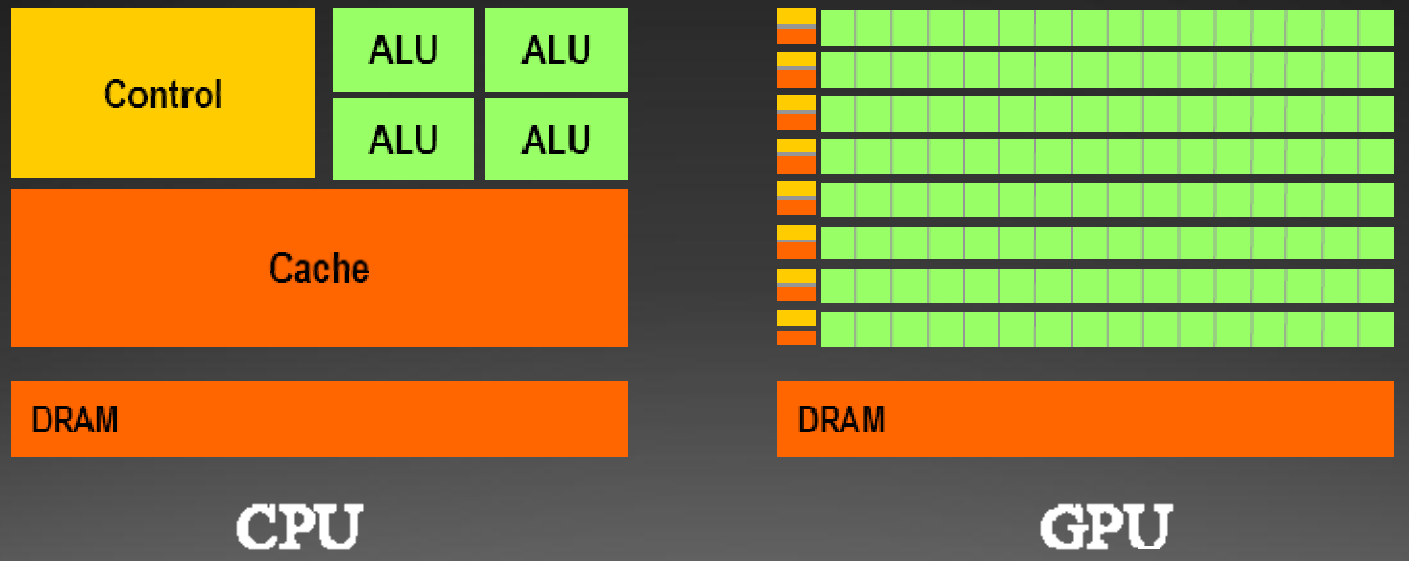
Why use GPU?

- 연산속도 : 367 GFLOPS vs. 32 GFLOPS
- 메모리 대역폭 : 86.4 GB/s vs. 8.4 GB/s

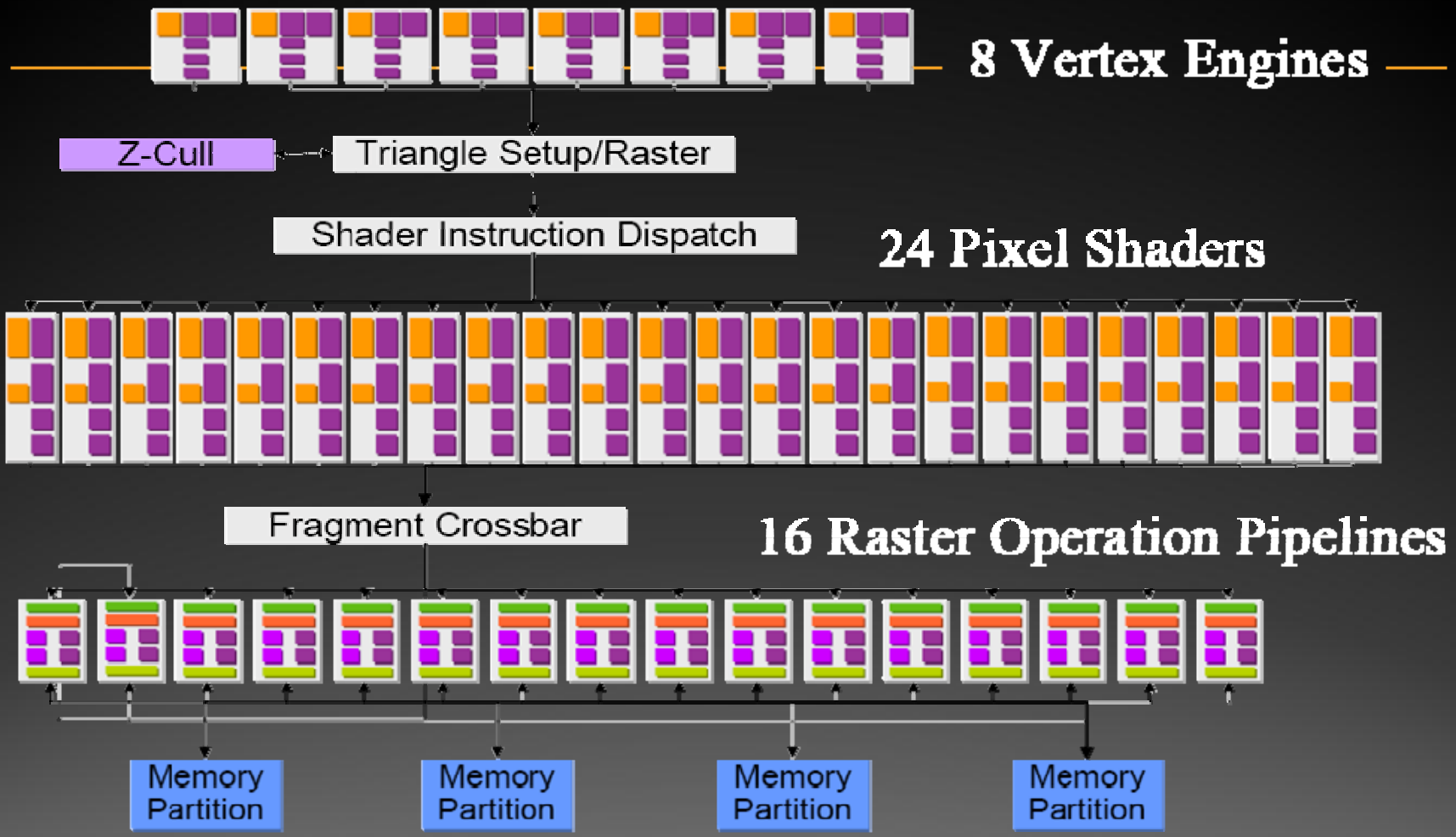


Why so fast GPU?

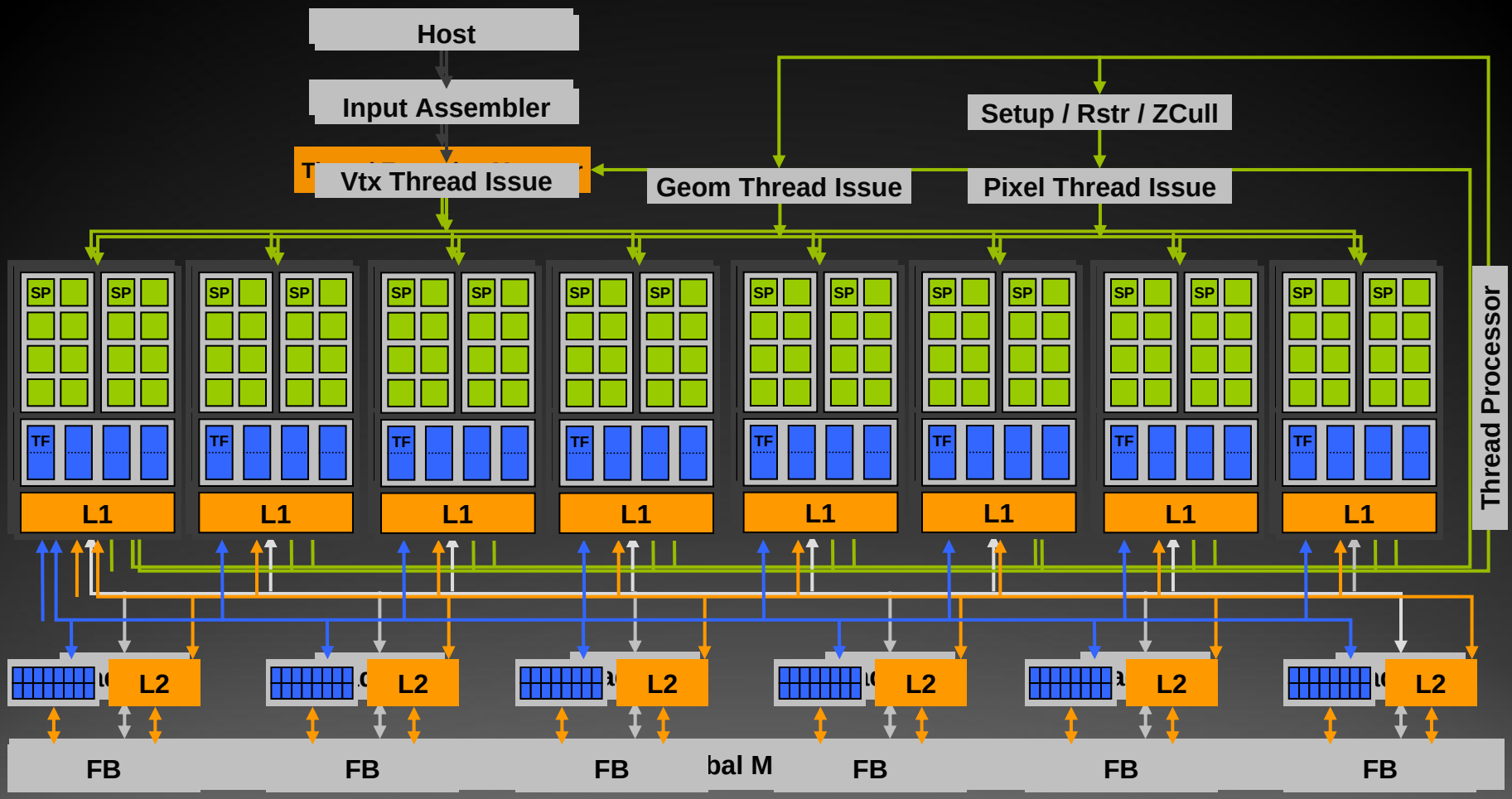
- GPU는 병렬처리와 수학적 연산에 대하여 특화
- 더 많은 트랜지스터가 흐름제어나 데이터 캐싱보다 데이터 처리에 집중



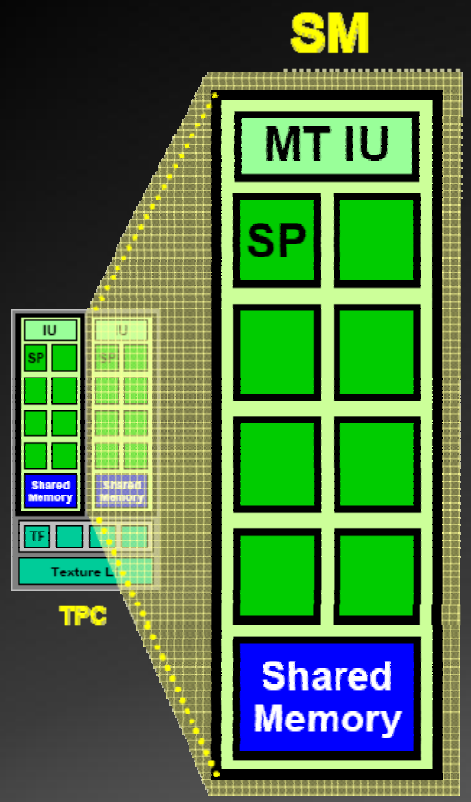
Geforce 7800 GTX



Geforce 8800 GTX



Streaming Multiprocessor



- 8개의 스트리밍 프로세서로 구성
- Load/store 구조
- 32-bit integer 명령어
- IEEE754 32-bit floating point
- Branch, call, return, predication
- 8K registers, 쓰레드에 배치
- 16K Shared memory
  - 협력하는 쓰레드간의 데이터 공유

What is “CUDA”?

- 『Compute Unified Device Architecture』
- 2007/02 CUDA Beta 최초 공개
- 2007/07 CUDA 1.0 정식버전 공개
- 2007/11 CUDA 1.1 공개
- 현재 CUDA 2.0 Beta 공개

What is “CUDA”? (continue.)

- General purpose programming model
  - 유저가 GPU의 쓰레드를 배치
  - GPU = 대용량 병렬처리 프로세서
- GPU에 프로그램을 적재하기 위한 드라이버
  - Driver Optimized for computation
  - 그래픽 API를 사용하지 않음
  - OpenGL 버퍼와 데이터 공유
  - 최고의 다운로드 & 다시읽기 속도 보장
  - 명백한 GPU 메모리 관리

CUDA Advantages over Legacy GPGPU

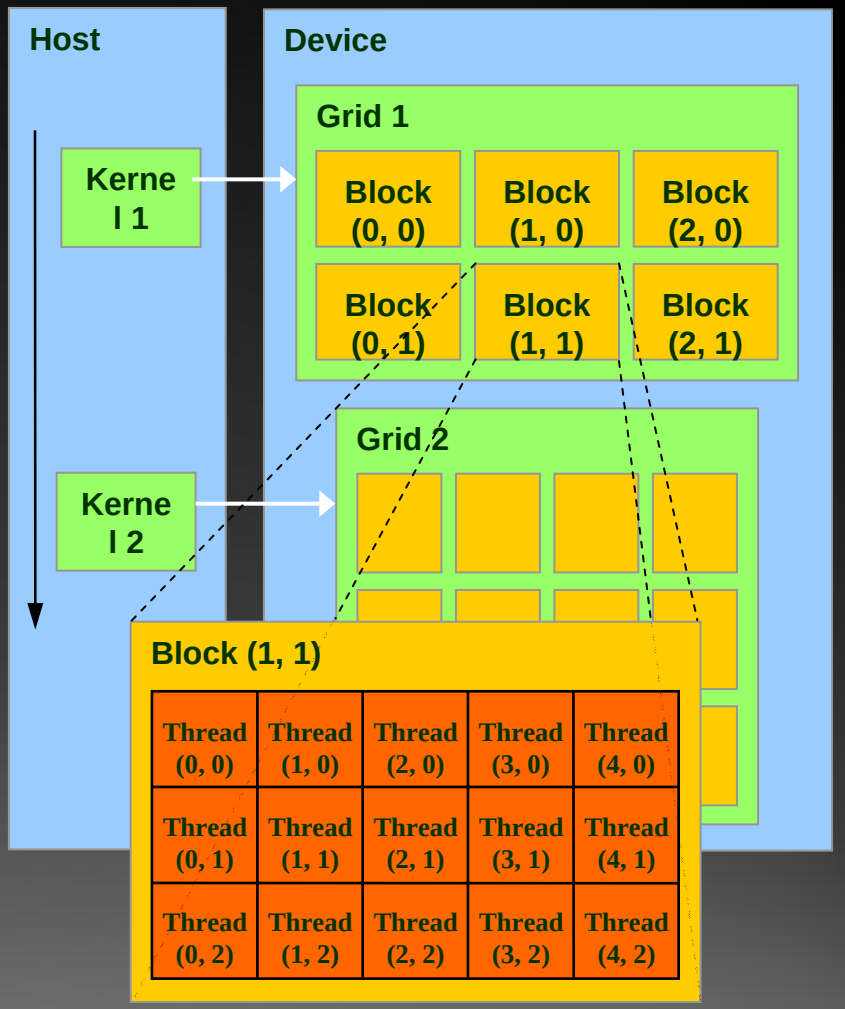
- 제한 없는 메모리 접근
  - 쓰레드는 필요한 부분에 읽고 쓰기 가능
- 공유메모리와 쓰레드
  - 쓰레드는 공유메모리에서 데이터를 읽어 협력가능
  - 한블럭안의 쓰레드는 누구나 공유메모리 접근가능
- 상대적으로 적은 지식 필요
  - C의 확장형 언어
  - 그래픽적인 지식 필요 없음
- 그래픽API에 의한 오버헤드가 없음

Environment

- Geforce 8시리즈 이상의 그래픽 카드
- Windows
  - Visual Studio 2003 or 2005
  - Visual Studio 2008 4분기 지원예정
- Linux
  - Redhat Enterprise Linux3.x, 4.x or 5.x
  - SUSE Linux Enterprise Desktop 10-SP1
  - OpenSUSE 10.1 or 10.2
  - Ubuntu 7.04
- Mac
  - OS X 10.5.2

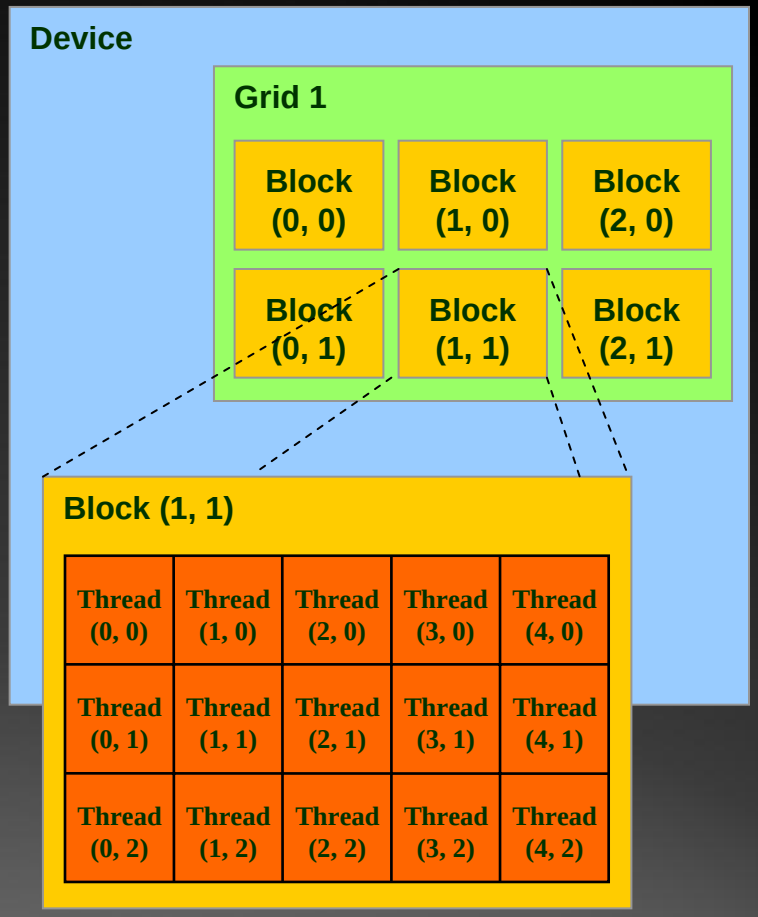
CUDA Programming Model

- Kernel = GPU 프로그램
- Grid = 커널을 수행하는 쓰레드 블록의 배열
- Thread block = 커널을 수행하고 공유메모리를 통하여 대화하는SIMD 쓰레드의 그룹



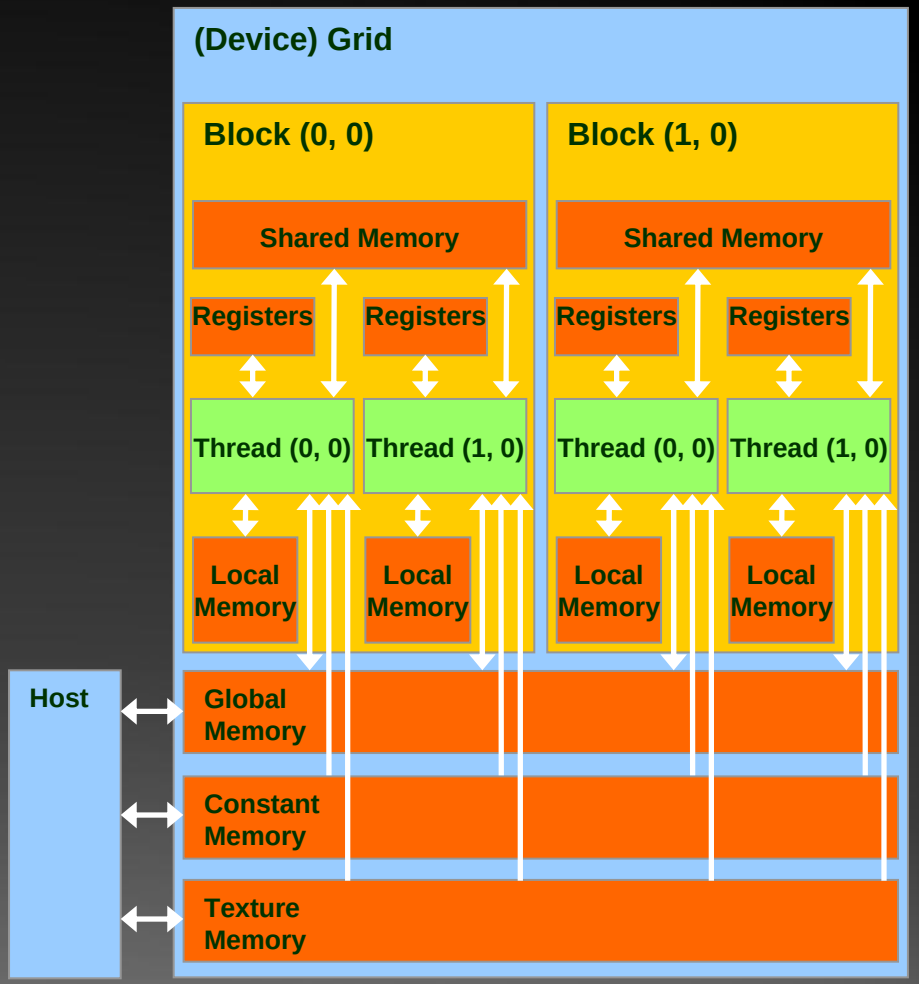
Thread and Block IDs

- 쓰레드와 블록은 ID를 가짐
  - 각각의 쓰레드는 일을 하기위한 데이터를 결정 가능
- Block ID : 1D or 2D
- Thread ID : 1D, 2D, or 3D
- 다차원의 데이터 처리는 메모리 참조를 간편하게 함
  - 이미지 프로세싱
  - 볼륨에서 편미분방정식 해결
  - etc.



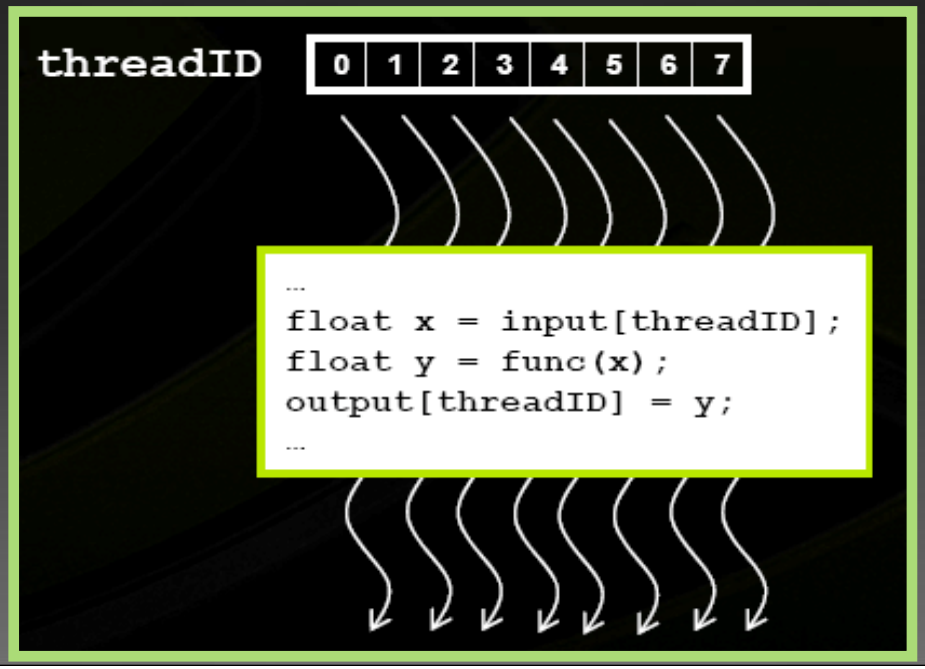
CUDA Memory Spaces

- 메모리 영역별 권한
  - Register : R/W 쓰레드
  - Local Memory : R/W 쓰레드
  - Shared Memory : R/W 블록
  - Global Memory : R/W grid
  - Constant Memory : R/O grid
  - Texture Memory : R/O grid
- Host
  - Global, Constant, and Texture Memory : R/W



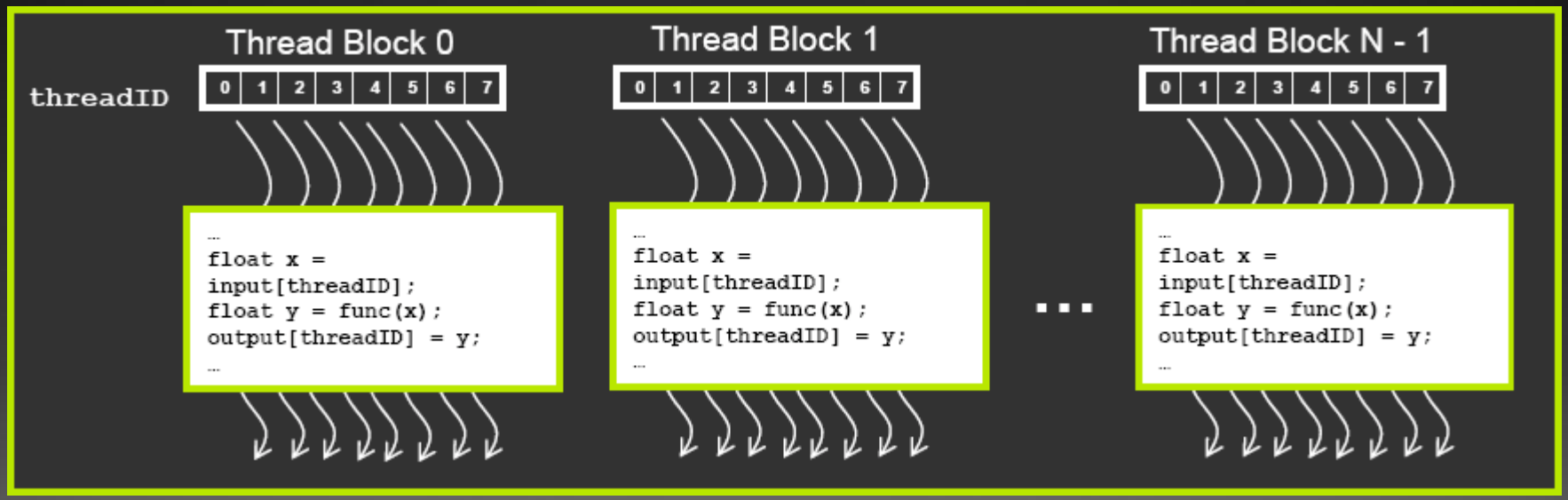
Arrays of parallel threads

- Kernel은 쓰레드의 배열에 의해 수행됨
  - 모든 쓰레드는 동일한 코드를 수행
  - 각각의 쓰레드는 컨트롤을 결정하고 메모리주소를 계산하기 위해 ID를 가짐



Thread Blocks

- 여러 개의 블록 안에 단일화된 쓰레드 배열로 나눠짐
  - 공유 메모리를 통하여 한 블록 안의 쓰레드가 협력
  - 다른 블록에 존재하는 쓰레드와 는 협력불가



Examples

- Increment Array Elements
- Neural Networks

Example : Increment Array elements

CPU program

```
void increment_cpu(float *a, float b, int N){
 for (int idx = 0; idx<N; idx++)
    a[idx] = a[idx] + b;
}
void main(){
    .....
    increment_cpu(a, b, N);
}
```

CUDA Program

```
__global__ void increment_gpu(float *a, float b, int N){

int idx = blockIdx.x * blockDim.x + threadIdx.x;
if (idx < N)
    a[idx] = a[idx] + b;
}

void main(){
.....
dim3 dimBlock (blocksize);
dim3 dimGrid( ceil( N / (float)blocksize) );
increment_gpu<<<dimGrid, dimBlock>>>(a, b, N);
}
```

Example : Increment Array Elements

N개의 요소를 가진 벡터a에 b 더하기

N=16, blockDim = 4라고 가정하면

blockIdx.x=0
blockDim.x=4
threadIdx.x=0,1,2,3
idx=0,1,2,3

blockIdx.x=1
blockDim.x=4
threadIdx.x=0,1,2,3
idx=4,5,6,7

blockIdx.x=2
blockDim.x=4
threadIdx.x=0,1,2,3
idx=8,9,10,11

blockIdx.x=3
blockDim.x=4
threadIdx.x=0,1,2,3
idx=12,13,14,15

```
int idx = blockDim.x * blockIdx.x + threadIdx.x;
```

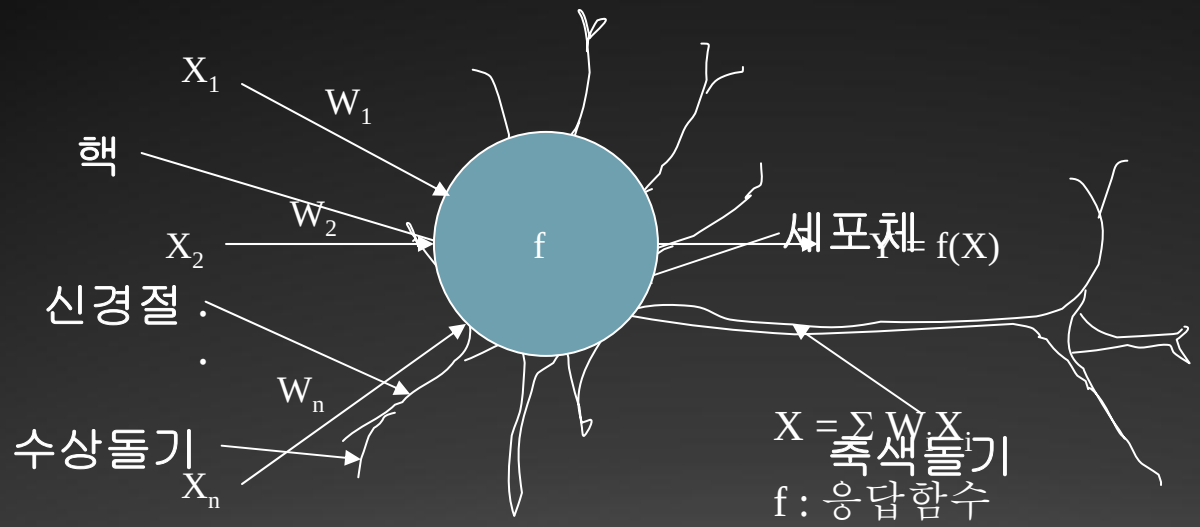
Neural Networks

- (인공) 신경회로망
  - 인간의 두뇌작용을 신경 세포들간의 연결관계로 모델링
    → 인간의 학습을 모델링
- 기본 작업
  - 학습(learning): 패턴 부류에 따라 신경망의 연결가중치 조정
  - 재생(recall): 학습된 가중치와 입력벡터와의 거리 계산하여 가장 가까운 클래스로 분류

  → 사람과 같은 학습 능력: 패턴 분류, 인식, 최적화, 예측
- 응용분야
  - 화상인식, 음성인식, 로봇제어 등 다양한 인공지능 분야에 적용

Concept

- 생물학적 뉴런



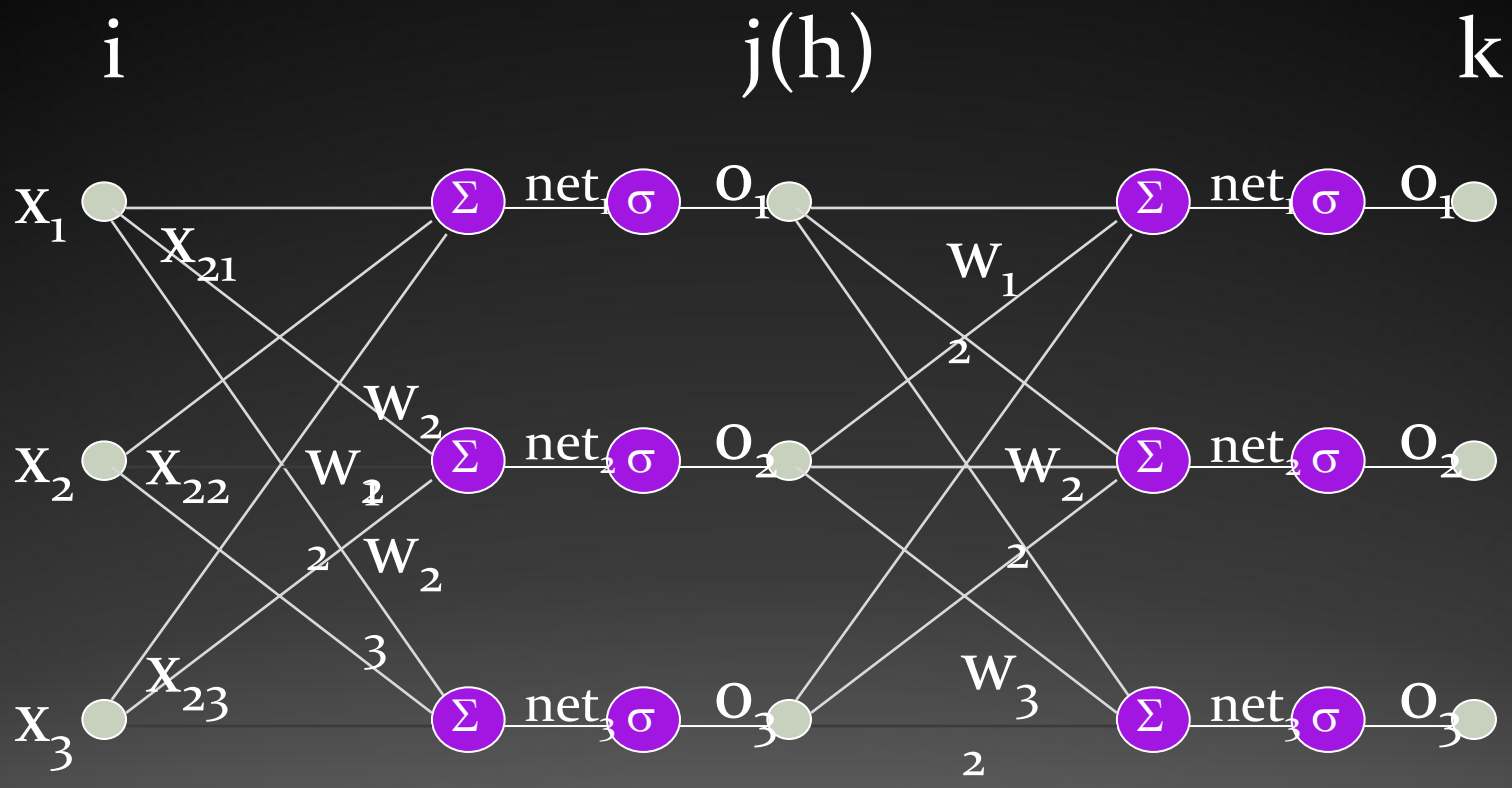
Kinds of Neural Networks

입력형식	학습방식	신경회로망 모델
이진입력	지도 학습	Hopfield network
	지도 학습 및 비지도 학습을 결합한 학습	Counterpropagation network
	비지도 학습	ART model
실수입력	지도 학습	Perceptron Multilayer Perceptron
	비지도 학습	Competitive learning SOM

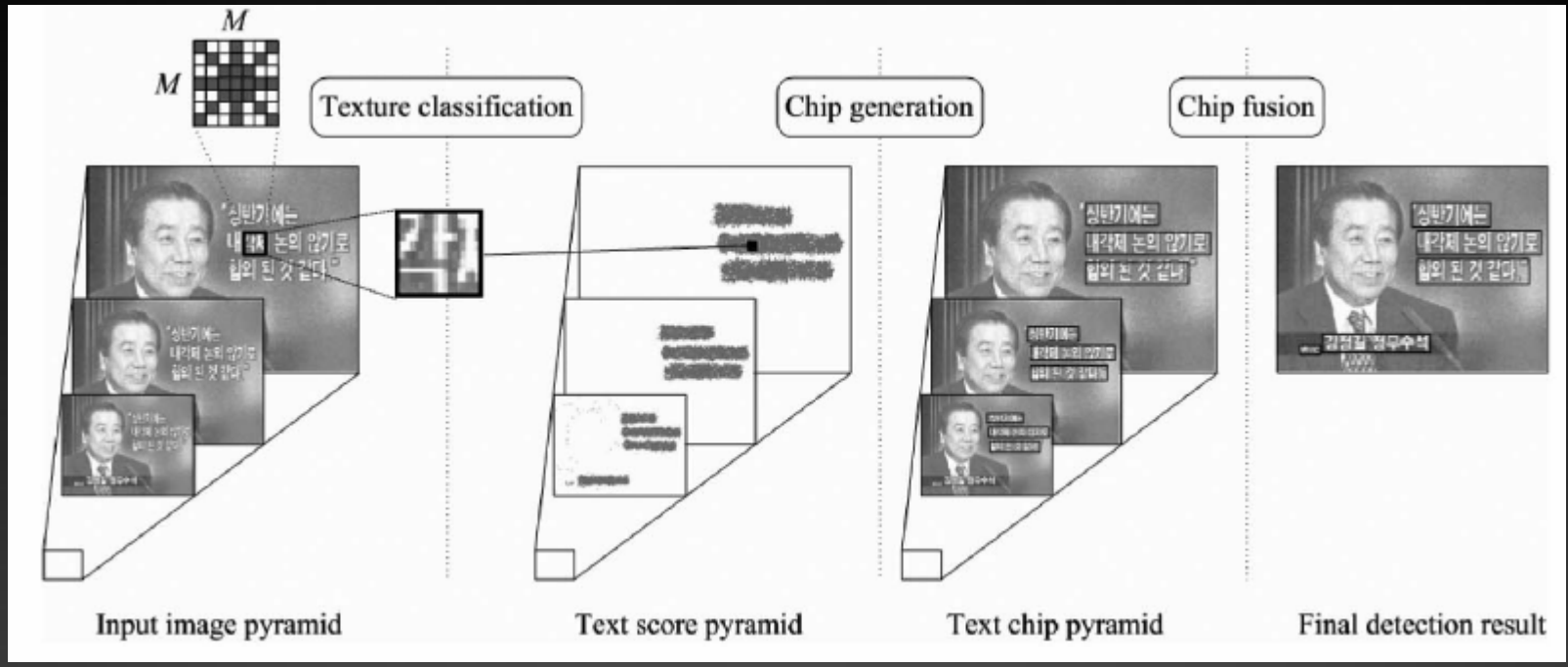
Multilayer Perceptron

- 1969년 Minsky's attack 이후 신경망 연구 침체
- 1986년 이후 Rumelhart 등의 연구에 의해 다층 신경망의 error backpropagation 알고리즘 완성
  - 비선형(non-linear) 문제를 학습 가능
  - 신경망 연구의 비약적 발전

Multilayer Perceptron Model



What do I say?


M
M
Texture classification
Chip generation
Chip fusion
Input image pyramid
Text score pyramid
Text chip pyramid
Final detection result

GPU Operation

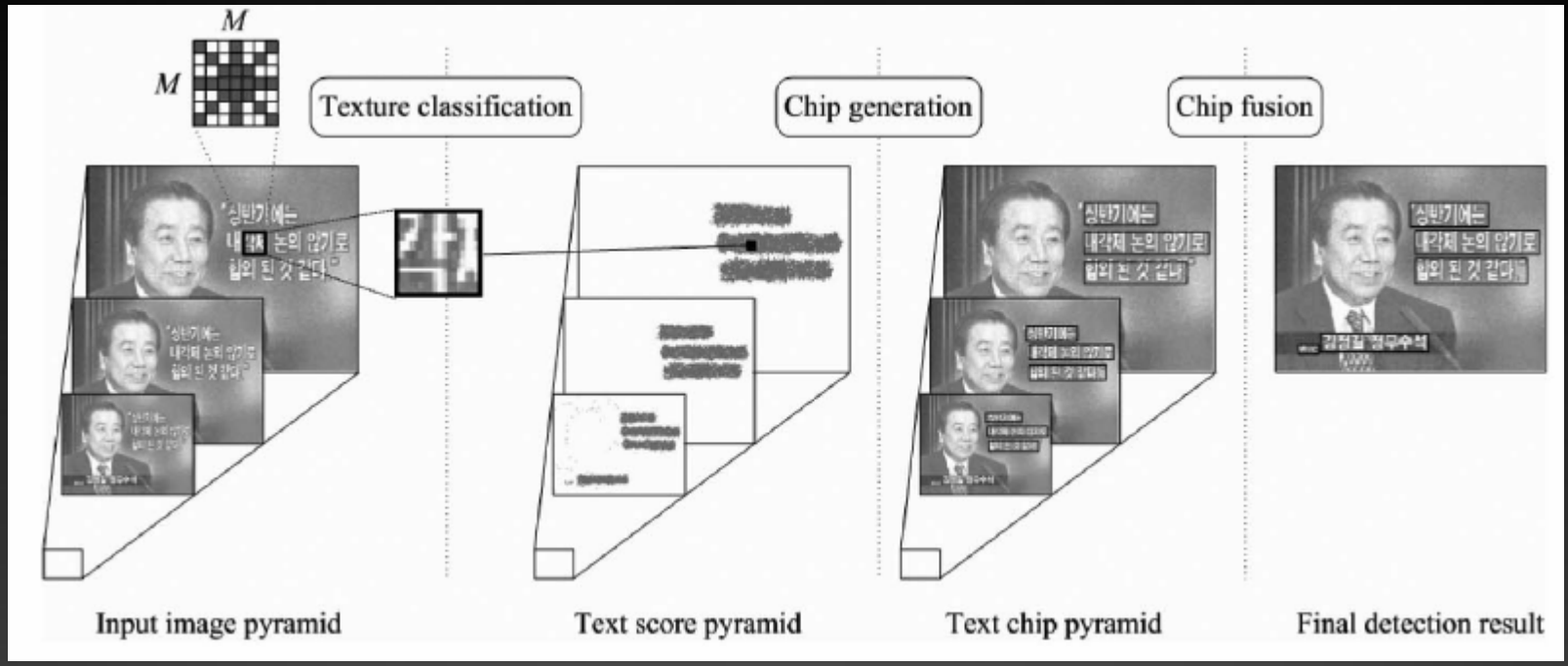
- 각 '노드'에서의 내적연산은 입력벡터와 웨이트벡터를 축적함으로써 행렬의 곱 연산으로 변환 가능

$$W=\begin{bmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1N} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2N} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ w_{M1} & w_{M2} & w_{M3} & \cdots & w_{MN} \end{bmatrix}=\begin{bmatrix} W_1 \\ W_2 \\ \cdots \\ W_M \end{bmatrix}, \quad X=\begin{bmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1L} \\ x_{21} & x_{22} & x_{23} & \cdots & x_{2L} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{N1} & x_{N2} & x_{N3} & \cdots & x_{NL} \end{bmatrix}=\begin{bmatrix} X_1^T \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} \quad B=\begin{bmatrix} b_1 & b_1 & b_1 & \cdots & b_1 \\ b_2 & b_2 & b_2 & \cdots & b_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ b_M & b_M & b_M & \cdots & b_M \end{bmatrix}$$

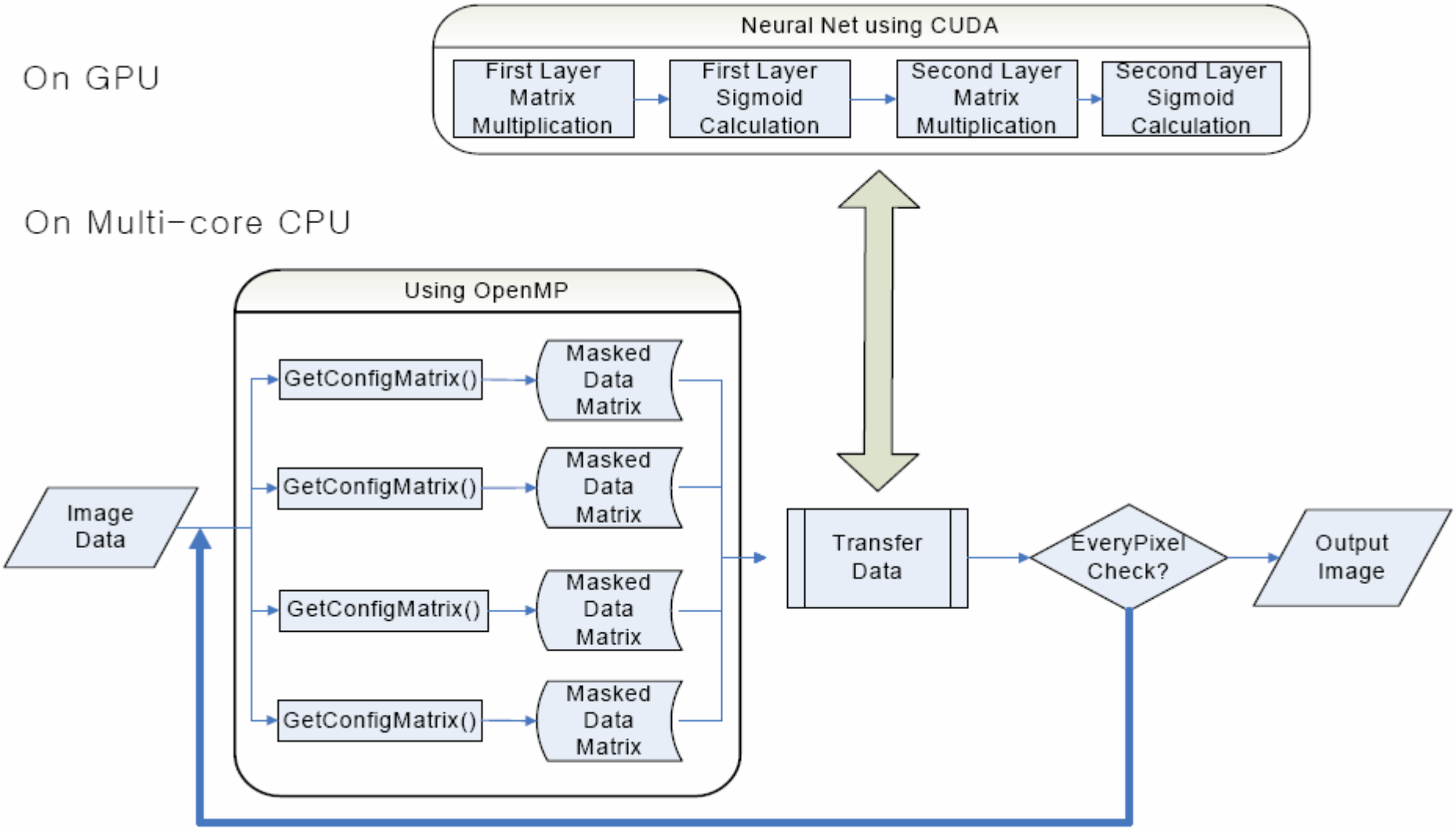
$$M=W\times X+B=\begin{bmatrix} W_1\cdot X_1 & W_1\cdot X_2 & W_1\cdot X_3 & \cdots & W_1\cdot X_N \\ W_2\cdot X_1 & W_2\cdot X_2 & W_2\cdot X_3 & \cdots & W_2\cdot X_N \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ W_M\cdot X_1 & W_M\cdot X_2 & W_M\cdot X_3 & \cdots & W_M\cdot X_N \end{bmatrix}+\begin{bmatrix} b_1 & b_1 & b_1 & \cdots & b_1 \\ b_2 & b_2 & b_2 & \cdots & b_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ b_M & b_M & b_M & \cdots & b_M \end{bmatrix}=\begin{bmatrix} m_{11} & m_{11} & m_{11} & \cdots & m_{11} \\ m_{12} & m_{12} & m_{12} & \cdots & m_{12} \\ m_{13} & m_{13} & m_{13} & \cdots & m_{13} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ m_{1M} & m_{12} & m_{12} & \cdots & m_{12} \end{bmatrix}$$

$$R=sigmoid(M)=\begin{bmatrix} (1+e^{-m_{11}})^{-1} & (1+e^{-m_{12}})^{-1} & (1+e^{-m_{13}})^{-1} & \cdots & (1+e^{-m_{1L}})^{-1} \\ (1+e^{-m_{21}})^{-1} & (1+e^{-m_{22}})^{-1} & (1+e^{-m_{23}})^{-1} & \cdots & (1+e^{-m_{2L}})^{-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ (1+e^{-m_{M1}})^{-1} & (1+e^{-m_{M2}})^{-1} & (1+e^{-m_{M3}})^{-1} & \cdots & (1+e^{-m_{ML}})^{-1} \end{bmatrix}$$

MLP Implementation



MLP Implementation



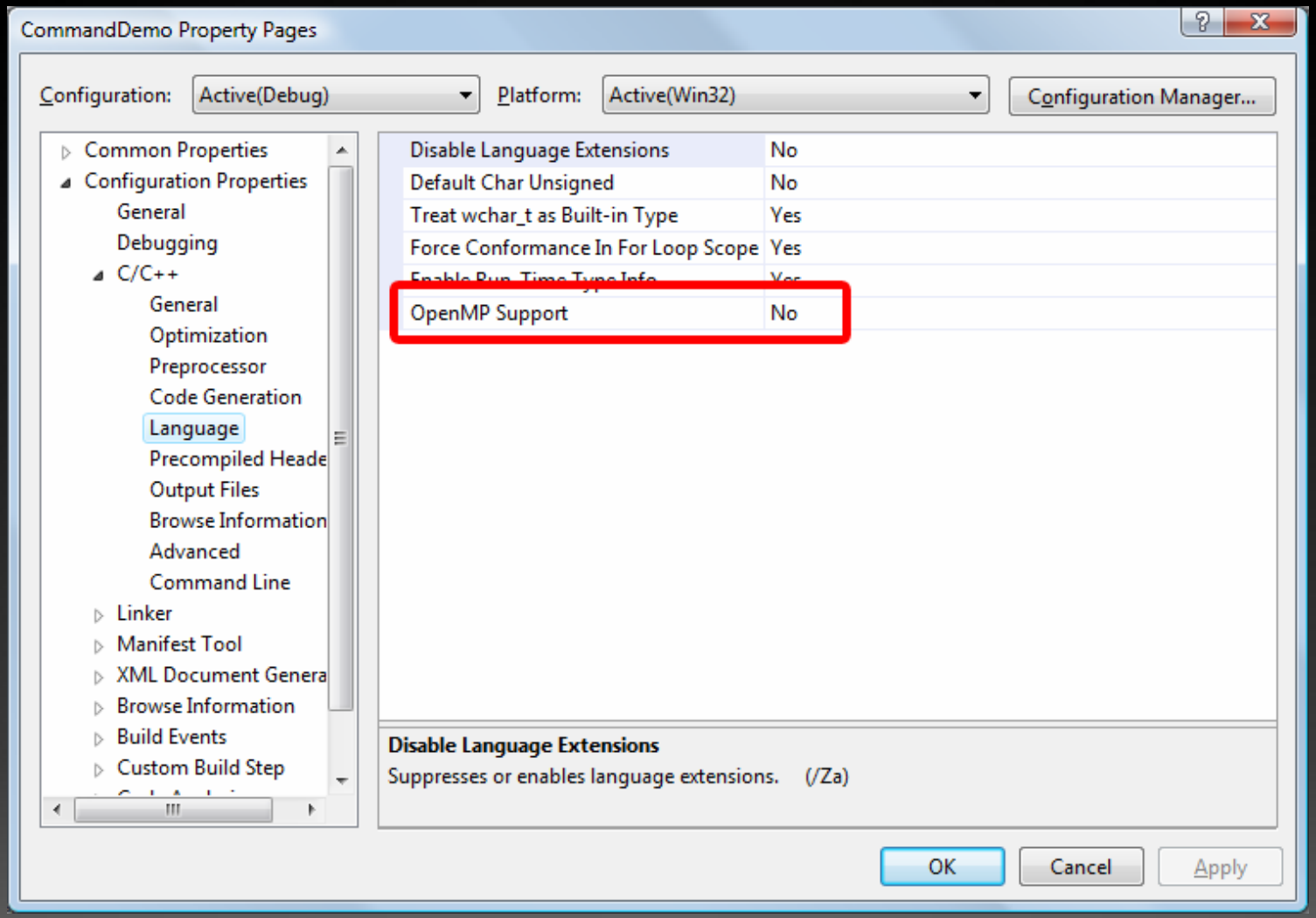
What is “OpenMP”?

- 『Open Multi-Processing』
- 1997/08 FORTRAN 1.0 표준 발표
- 1998/08 C/C++ 1.0 표준 발표
- 2002/03 FORTRAN, C/C++ 2.0 표준 발표
- 2005/05 C/C++/FORTRAN 통합 2.5 표준 발표

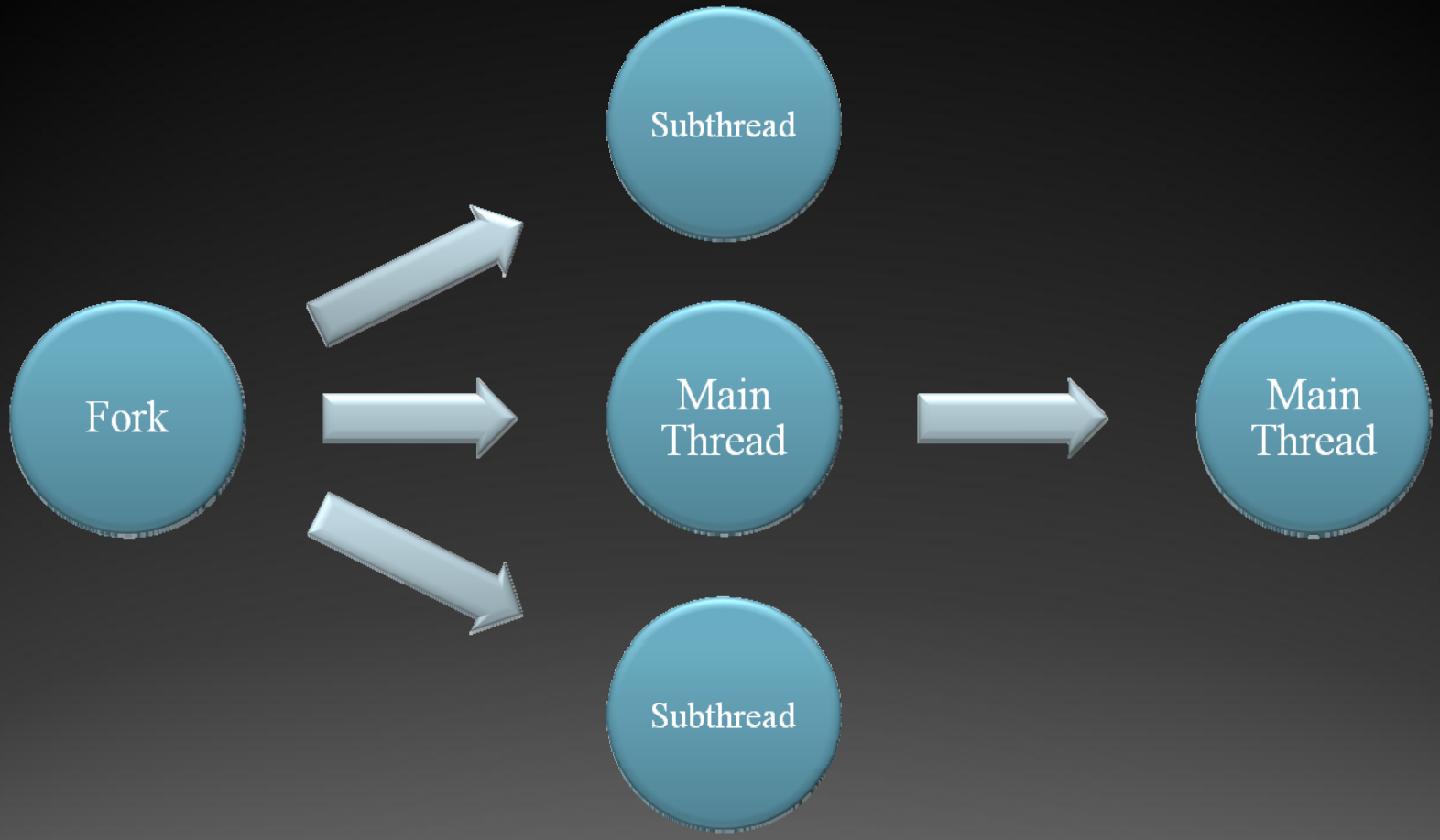
Why OpenMP?

- GPU로의 잦은 데이터 전송에 의한 지연발생
- 대용량의 데이터를 생성하여 데이터 전송
- 대용량의 데이터 생성에 오버헤드 발생
- 멀티코어 cpu를 이용한 병렬처리로 오버헤드 감소
- 소스의 적은 수정으로 병렬처리 가능

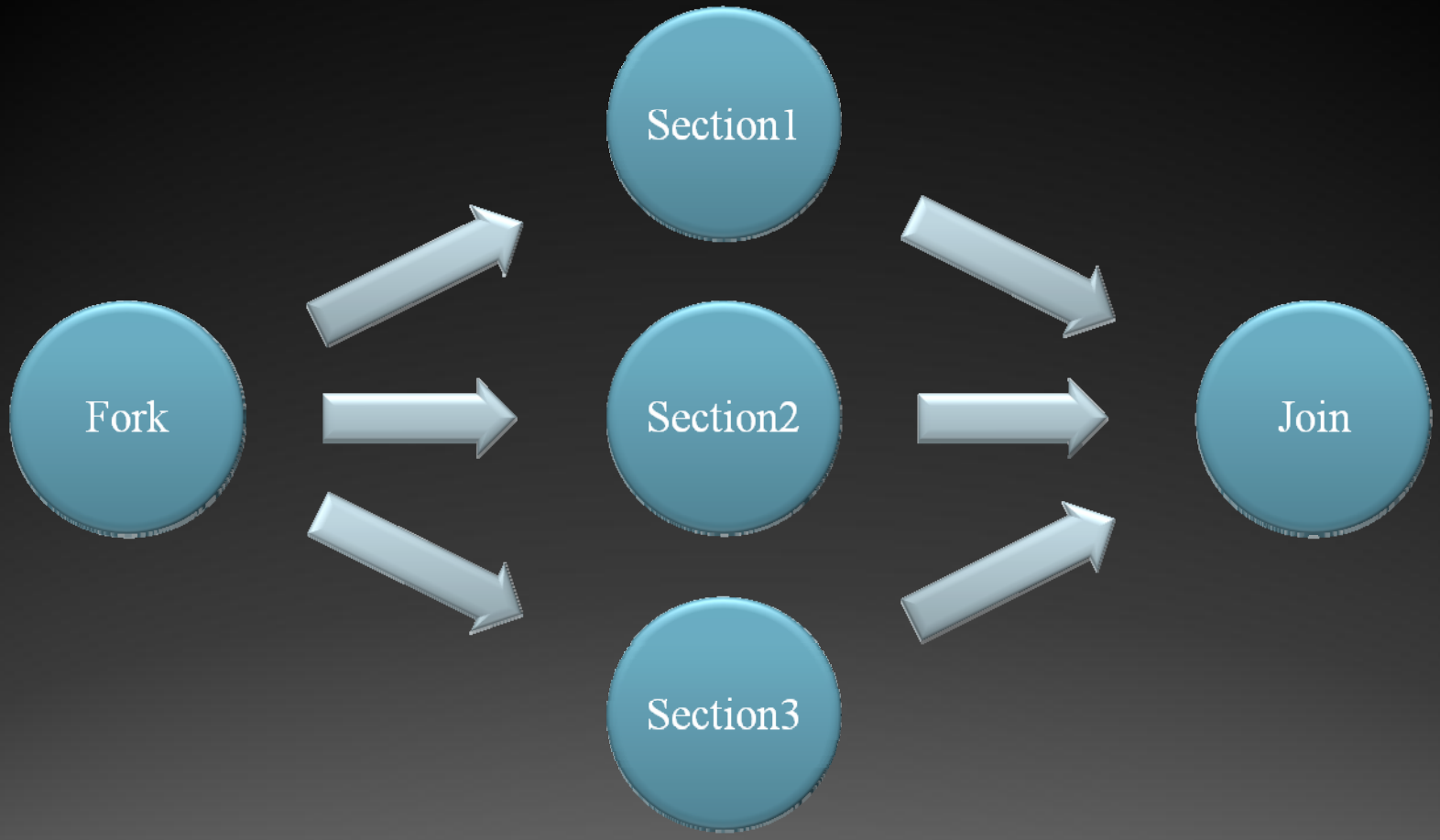
OpenMP Preference (later VS2005)



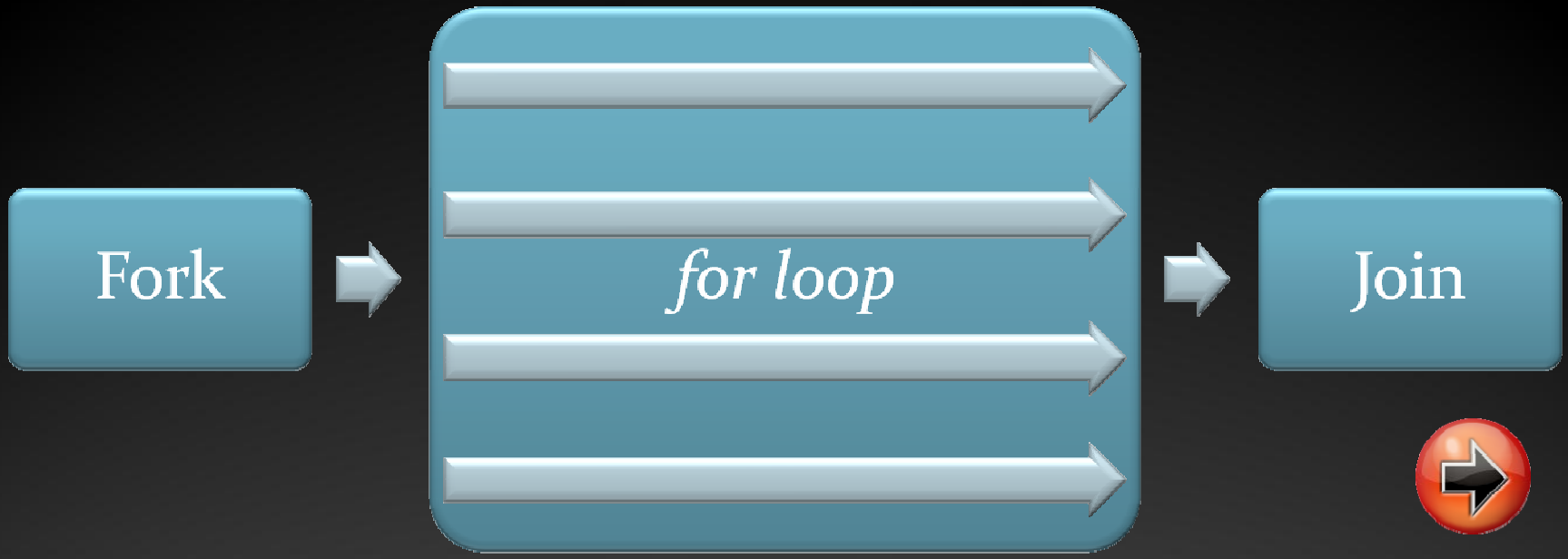
Thread Model



Fork-Join Model

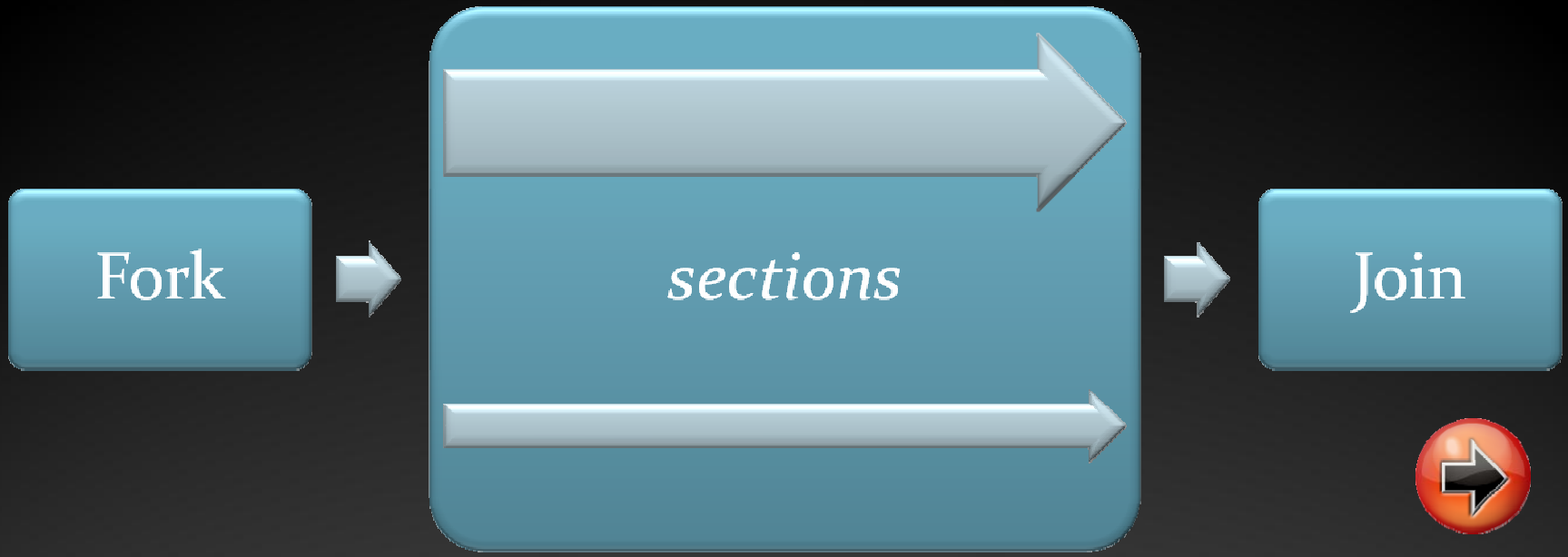


Parallel Section(for)




```
#pragma omp for [clause [clause ...] ]
{
   for loop
}
```

Parallel Section(sections)



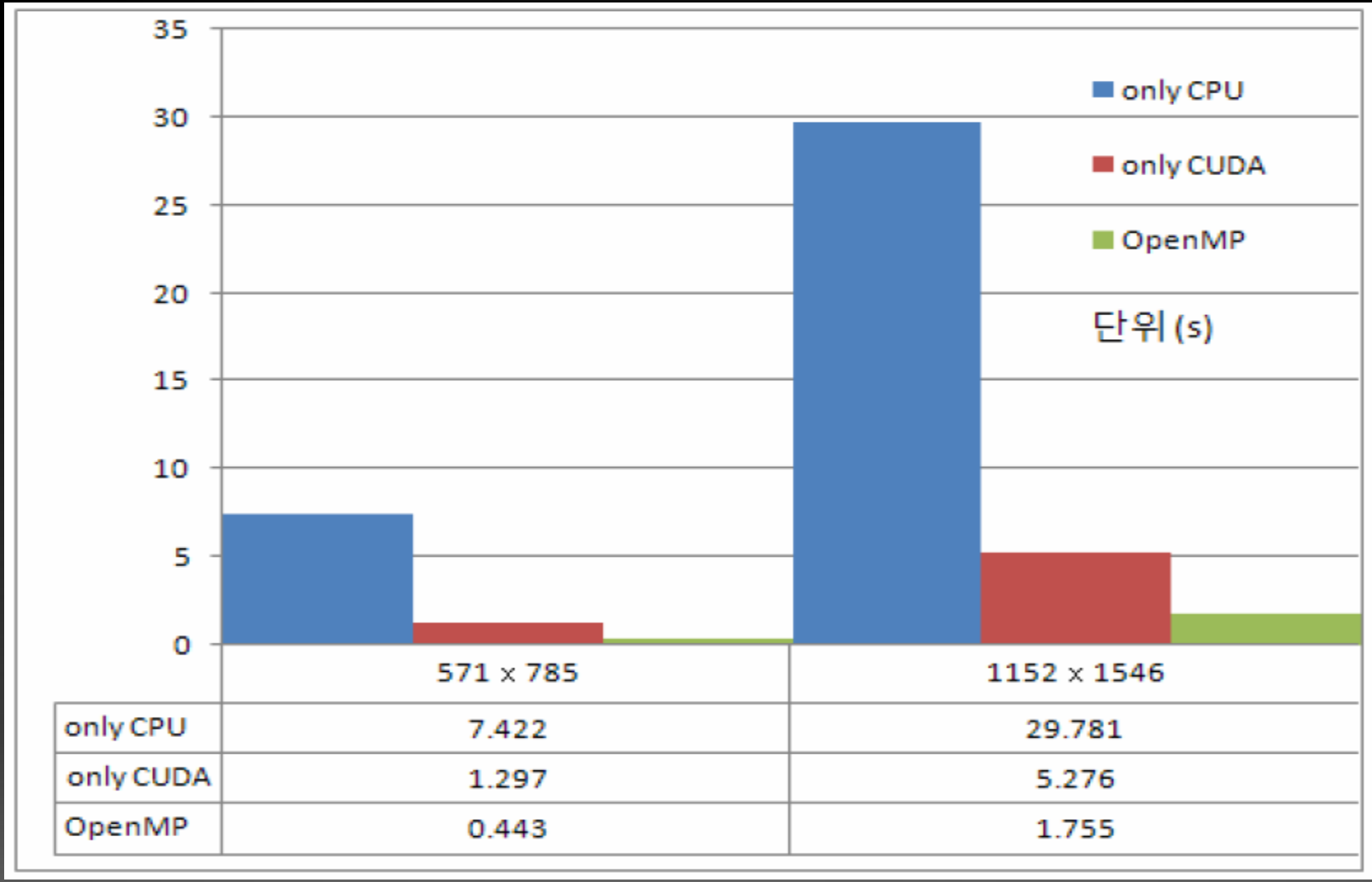

```
#pragma  omp  sections [clause [clause …] ] {
   [#progma  omp  section]
      structured code block
}
```

Using Neural Network

```
#pragma omp parallel sections{
        #pragma omp section{
                int y1 = y + count*INPUT_WIDTH/(mWidth-10);
                int x1 = x + (count*INPUT_WIDTH)%(mWidth-10);
                GetConfigMatrix( x1, y1, input1);
        }
        … 중략 …
        #pragma omp section{
                int y4 = y + (count+3)*INPUT_WIDTH/(mWidth-10);
                int x4 = x + ((count+3)*INPUT_WIDTH)%(mWidth-
10);
                GetConfigMatrix(x4, y4, input4);
        }
}
```

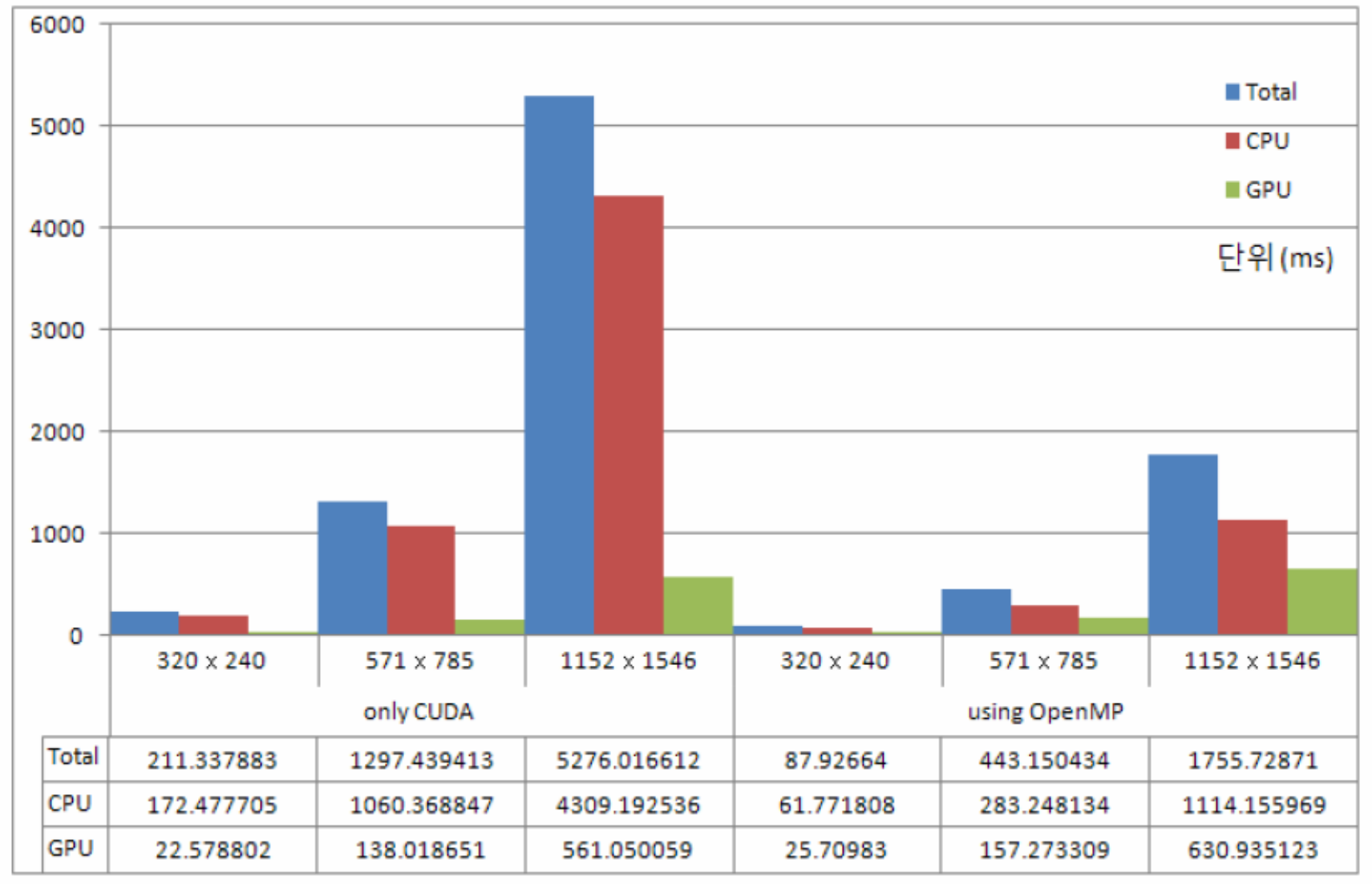
Experimental Results

Time Complexity



	571 x 785	1152 x 1546
only CPU	7.422	29.781
only CUDA	1.297	5.276
OpenMP	0.443	1.755

Time Complexity



	320 x 240	571 x 785	1152 x 1546	320 x 240	571 x 785	1152 x 1546
	only CUDA			using OpenMP		
Total	211.337883	1297.439413	5276.016612	87.92664	443.150434	1755.72871
CPU	172.477705	1060.368847	4309.192536	61.771808	283.248134	1114.155969
GPU	22.578802	138.018651	561.050059	25.70983	157.273309	630.935123

참고논문

- GPU implementation of neural networks , International Journal of Pattern Recognition, Vol. 37, Issue 6, Pages 1311-1314, 2004, **SCIE**

- CUDA와 OpenMP를 이용한 신경망 구현, Korea Computer Congress 2008(2008 한국컴퓨터종합학술대회), 발표예정