고급 컴퓨터 구조 과제2

병렬 처리 컴퓨터 구조에서의 Game Of Life 구현과 분석

201521032 한태희

0. 보고서의 목적

이번 과제의 목적은 콘웨이의 생명 게임을 순차적으로 구현하거나 병렬적으로 구현해, 소프트웨어의 병렬적 구현 방식과 그것을 작동시키는 하드웨어가 소프트웨어의 속도와 성능에 어떤 영향을 미치는지 알아보는 것이다. 이를 위해서 순차적으로 작동하는 코드, pThread를 사용해 병렬처리를 진행하는 코드, 그리고 마지막으로 CUDA를 이용해 GPU가 병렬처리를 진행하는 코드 3가지 종류를 구현했다.

그 뒤, 구현한 3가지 종류의 생명 게임의 작동 특징을 분석했다. 프로세스의 결과로 출력되는 것은 프로세스가 작동하는데 소모된 시간이다. 그리고 이 소모 시간을 통해서,

(전체 cell 의 개수 / 소모된 시간) = cell/sec (시간당 셀 처리율)

이란 단위를 구해, 프로세스의 성능을 평가했다.

1. 디자인 및 구현 요약

본 과제는 실습용으로 제공된 CUDA 환경 지원 리눅스 서버 컴퓨터에서 vi를 이용하여 작성, 실행되었다. 코드 양식은 제공된 스켈레톤 코드와 명시된 입출력, 자료구조에 맞추어 작성되었다. 싱글코어에서 순차적으로 작동하는 버전, pThread로 병렬 처리하는 버전, GPU로 병렬처리 하는 버전 3가지 모두 구현이 완료되었다.

콘웨이의 생명게임을 pThread와 CUDA 환경에 맞게 생명게임을 병렬 처리하기 위해선, 아래 2가지 절차를 반복하면 된다.

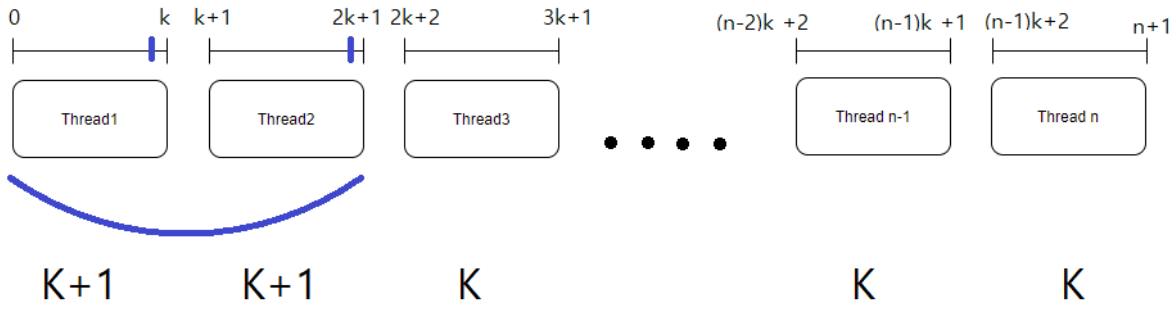
1. 각 병렬 코어는 자신이 할당 받은 from(시작) 좌표부터 to(끝) 좌표까지 읽기 전용 그리드를 읽음. cell의 다음 단계 생사 여부를 쓰기 전용 그리드에 기록. (싱글 코어의 작동과 동일)
2. 모든 병렬 코어의 작동이 끝나면, 비병렬 제어부가 읽기 전용 그리드와 쓰기 전용 그리드의 역할을 서로 바꿈. 이를 통해 다음 반복 준비 완료.

위의 작동을 하기 위해선, 병렬 처리 장치 하나당 할당되는 from과 to의 범위를 균등하게 정의해

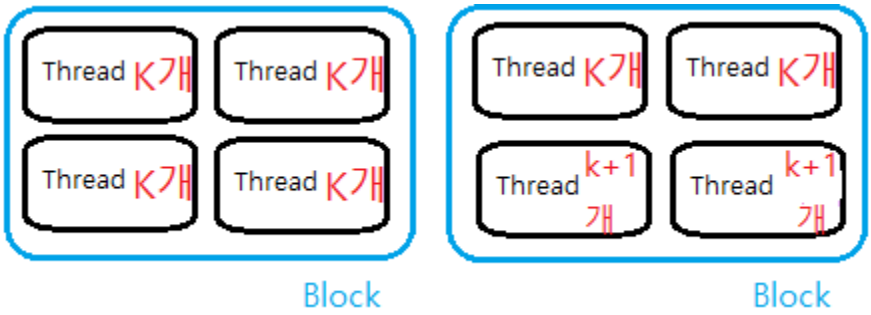
줄 필요가 있다. 만약 from과 to를 균등하게 할당하지 않는다면, (모든 병렬 코어가 동시에 작동한다는 가정 하에) 하나의 병렬 코어가 다른 병렬 코어보다 더 많은 시간을 소모할 것이다. 이는 병렬 처리에서의 병목 현상을 발생시킨다.

pthread의 경우, 전체 셀의 개수를 C, 스레드의 개수를 n이라고 한다면, 각 스레드는 C / n개의 셀을 분담하면 된다. 그리고 C % n개의 나머지는 일부 셀이 하나씩 추가로 더 작업하면 된다. C / n = k 라고 하면, 분배는 아래 그림과 같이 진행된다.

ex) 나머지가 2개일 경우

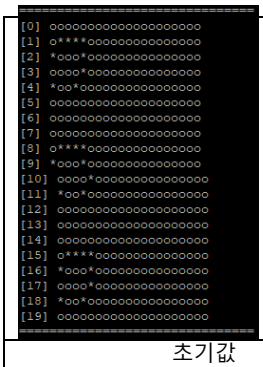


GPU (CUDA 언어)의 경우, m개의 블록과 하나의 블록당 n개의 스레드를 실행시키면, 총 m*n개의 스레드가 실행되게 된다. 따라서 논리적으로 코딩을 할 때는 m*n개의 스레드가 있다고 간주하고 병렬적으로 작업을 할당하면 된다. GPU 커널 함수는 자신을 실행하는 블록, 스레드 인덱스를 확인할 수 있다. 그것을 이용해 인덱스가 낮은 순서대로 k개의 블록을 스스로 할당하게 하면 된다. 그 뒤 남는 나머지는 하나씩 끝에 위치한 스레드들이 추가로 작업하게 하면 된다.

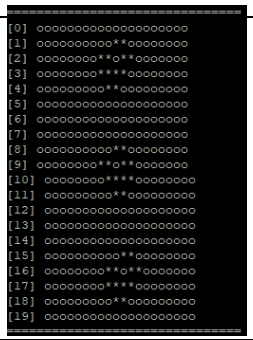


2. 결과 테스트

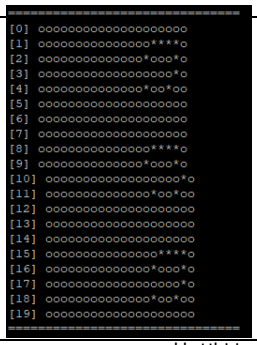
50*50 그리드에서 왼쪽 위의 글라이더가 오른쪽 아래로 이동하는 첫번째 테스트 케이스, 20*20 그리드에서 3개의 우주선이 왼쪽에서 오른쪽으로 이동하는 두번째 테스트 케이스를 통해 통해 전체 프로세스의 정상 작동을 블랙박스 테스트했다. 사진으로 첨부된 GPU 병렬처리 뿐만 아니라 순차적 구현과 pThread를 사용한 구현 모두 테스트 케이스에서 정상 작동함을 확인할 수 있었다.



초기값



15회 반복



28회 반복

위 사진들은 block=10, thread=10, GPU 환경에서 3개의 우주선이 오른쪽으로 이동하는 2번째 테스트 케이스를 확인해본 결과이다. 사진으로 첨부된 GPU 병렬처리 뿐만 아니라 순차적 구현과 pThread를 사용한 구현 모두 테스트 케이스에서 정상 작동함을 확인할 수 있었다.

3. 파라미터에 따른 출력 결과 변화 분석

3.1. 여러 작동 환경에서의 셀 처리 속도 비교

생명게임이 작동하는 환경은 그대로 둔 채, 생명게임이 진행되는 반복 횟수를 늘릴 경우, 반복 횟수와 소모 시간은 선형적으로 비례할 것이다. 고정된 작동 환경에서의 셀 처리 속도는 일정하기 때문이다. 따라서 반복 횟수를 늘려가며 소모 시간을 측정하면, 특정 환경에서의 셀 처리 속도를 회귀(기울기 구하기)를 통해 알아낼 수 있을 것이다.

생명 게임이 작동하는 환경은 5가지의 경우를 사용하였다.

Single Core	순차적인 작동 (nprocs =1)
10 pThreads	10개의 pThread 사용
500 pThreads	500개의 pThread 사용
100B, 10Th	cuda: 100개의 Block, 블록당 10개의 Thread
1000B, 1000Th	cuda: 1000개의 Block, 블록당 1000개의 Thread

그리고 Grid (Cell들의 집합)의 크기도 3가지의 경우를 사용하였다.

100*100 Grid (Cells), 1000*1000 Grid, 10000*10000 Grid

- 100*100 Grid의 경우

	0(회)	100	200	300	400	500	600	700	800	900	1000
Single Core	0(sec)	0.061	0.124	0.177	0.226	0.286	0.343	0.402	0.452	0.507	0.540
10 pThreads	0.000	0.062	0.124	0.192	0.253	0.305	0.369	0.441	0.490	0.549	0.625
500 pThreads	0.000	2.777	5.540	8.400	11.079	13.886	16.641	19.636	22.077	24.978	27.782

100B, 10Th	0.000	0.001	0.002	0.002	0.003	0.004	0.005	0.005	0.006	0.007	0.008
1000B, 1000Th	0.000	0.001	0.002	0.003	0.004	0.005	0.006	0.007	0.008	0.009	0.010

- 1000*1000 Grid의 경우

	0(회)	100	200	300	400	500	600	700	800	900	1000
Single Core	0(sec)	7.798	13.789	25.252	37.490	43.672	48.015	56.310	65.307	77.082	89.142
10 pThreads	0.000	1.508	3.030	4.482	5.980	7.505	9.015	10.527	11.993	13.626	15.066
500 pThreads	0.000	2.329	4.604	7.005	9.285	11.607	13.775	16.161	18.561	20.839	23.081
100B, 10Th	0.003	0.054	0.105	0.143	0.171	0.199	0.239	0.314	0.320	0.361	0.473
1000B, 1000Th	0.004	0.008	0.013	0.018	0.022	0.027	0.031	0.037	0.041	0.046	0.051

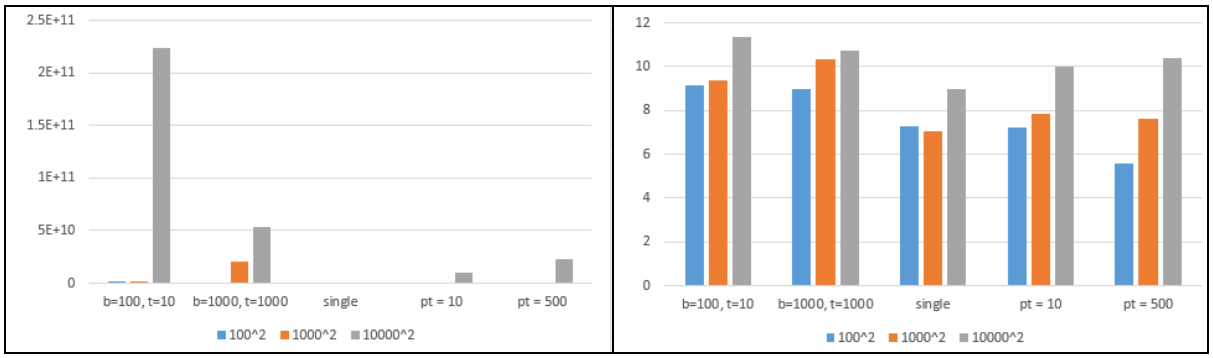
- 10000*10000 Grid의 경우

	0(회)	100	200	300
Single Core	0(sec)	1061.300	2119.960	3029.420
10 pThreads	0.000	105.814	208.216	314.102
500 pThreads	0.000	42.623	85.142	127.790
100B, 10Th	0.329	4.969	9.365	13.733
1000B, 1000Th	0.533	19.228	38.203	57.145

예상대로 모든 측정데이터는 반복 횟수에 비례하여 선형적으로 처리 시간이 증가하는 것을 확인할 수 있다. 회귀 분석을 사용해서 5가지 환경의 평균적인 셀 처리 속도를 구하면 아래와 같다.

	100^2	1000^2	10000^2
b=100, t=10	1.33474E+09(cell/sec)	2.32028E+09	2.24140E+11
b=1000, t=1000	9.81856E+08	2.11200E+10	5.29623E+10
single	1.81843E+07	1.14875E+07	9.84213E+08
pt = 10	1.61936E+07	6.63370E+07	9.57164E+09
pt = 500	3.60208E+05	4.32855E+07	2.34803E+10

속도 정보를 그래프로 표시하면 아래와 같이 나타낼 수 있다. 오른쪽의 그래프는 Cell/sec의 단위를 로그 스케일의 단위로 변환한 것이다.



그래프에서 확연히 나타나듯이, GPU를 사용하여 구현된 콘웨이의 생명 게임은 CPU를 사용했을 때보다 기하급수적으로 빨리진 작동 시간을 보여주었다. 가장 극적인 작동 속도 증가를 보인 것은 block=100, thread=10으로 GPU가 동작하고, 10000^2 크기의 그리드를 처리했을 때이다. 동일한 환경에서 순차적 방식의 프로세스가 약 1061초 걸린 작업을 block=100, thread=10 환경의 GPU는 약 5초만에 처리하는 것을 볼 수 있었다.

또다른 특징으로는 처리 데이터 양이 늘어나면 셀 처리 속도가 크게 증가하는 경향을 보였다는 것이다. 이는 CPU, GPU 양쪽의 아키텍처에서 공통으로 나타난 사항이다. 이는 2가지 원인이 있다고 생각된다. 첫번째는 사용된 데이터가 100^2과 1000^2의 경우엔 오랫동안 증식하는 콘웨이의 생명게임 예제를 사용한 반면에, 10000^2의 경우엔 전체가 공백으로 빈 데이터를 사용했기 때문일 것이다. 주변 셀 환경을 확인하는 작업은 if문을 사용한 조건절로 처리했기 때문에 전체 데이터가 빈 경우 셀 하나를 처리할 때 프로그램 카운터를 점프시킬 필요가 전혀 없어 처리속도가 증가했을 것이다. 두번째는 데이터의 크기가 커져 오랫동안 동일한 반복작업이 계속되기 때문에 컴퓨터 구조 내부의 캐시가 점점 최적화된 구조로 바뀌어 처리 속도가 증가했다고 생각된다.

CPU와 GPU의 실행 환경뿐만 아니라, 각각의 병렬처리 환경도 스레드(논리적 병렬처리 단위)의 개수에 따라서 실행속도가 차이나는 모습을 보여주었다. 다만 이는 혼란스러운 부분이 있었다. pThread의 경우엔 100^2의 데이터는 10개의 스레드가 빨랐지만, 1000^2의 데이터에선 비슷한 실행 속도를 보였고, 10000^2개의 데이터는 500개의 스레드가 더 빨라진 측정 결과를 보였다. GPU의 경우엔 100^2의 데이터는 두 환경이 비슷했지만 1000^2의 데이터는 b=1000, t=1000이 빨랐고 10000^2의 데이터는 b=100, t=10의 환경이 훨씬 빨랐다.

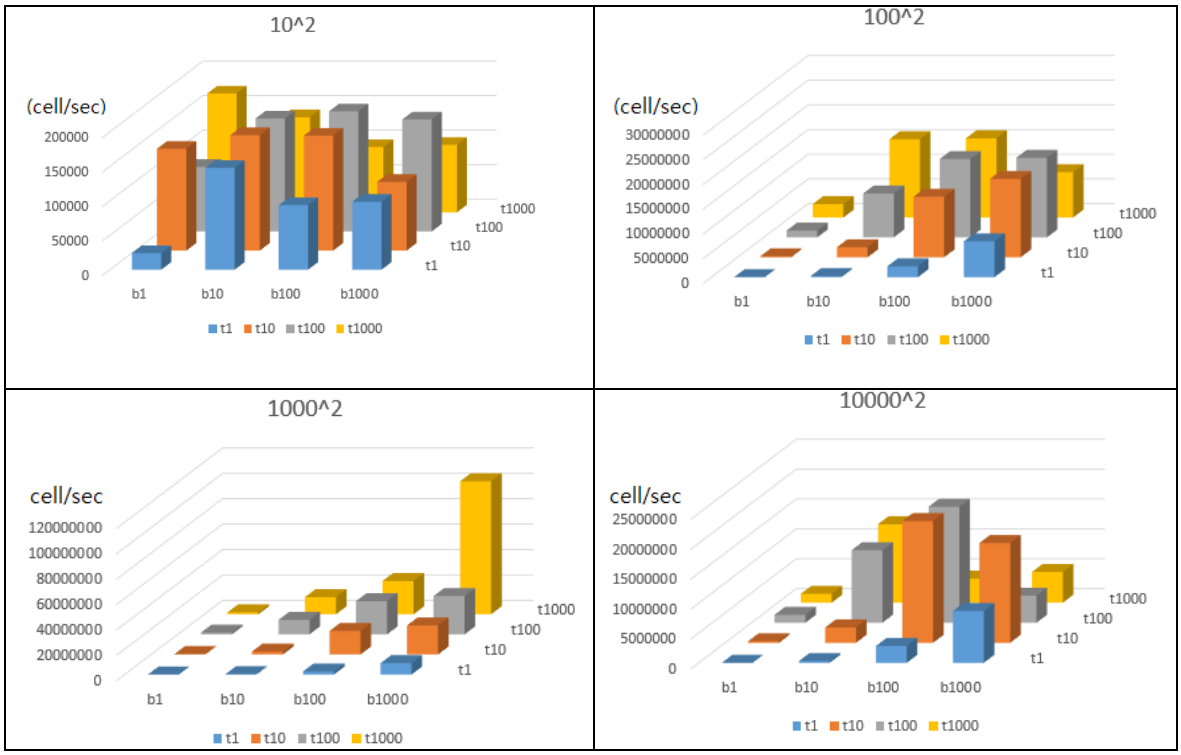
3.2. CUDA 환경에서 block 과 thread 개수에 따른 실행속도 비교

위의 궁금증을 해결하기 위해 block 과 thread의 인자 값을 바꾸어 그에 따른 실행 시간을 비교하는 것으로 관계를 밝혀보고자 했다. 콘웨이의 생명게임의 반복 횟수 증가에 따라서 처리 시간도 선형적으로 증가한다는, '처리 속도'가 존재한다는 것은 이미 확인했으므로 고정적으로 100회 반복만을 하고, 독립변수로 block 인자 값, thread 인자 값, 그리고 데이터의 크기를 변화시켰다.

cell size를 결과 시간으로 나눈 cell/sec 단위는 3.1. 에서 구했던 셀 처리량과 동일한 단위량이 되므로, 그것을 기준으로 측정을 하면 된다.

		10^2(cell size)	100^2	1000^2	10000^2
b1(block)	t1(thread)	24073.18 (cell/sec)	25290.59	29798.53	#DIV/0!
b1	t10	147710.5	214942.8	263613.7	262042.8
b1	t100	93720.71	1381788	1349473	1319535
b1	t1000	172413.8	2736727	1501190	1486839
b10	t1	147710.5	234984.5	291181	294109
b10	t10	167504.2	2040400	2258534	2573526
b10	t100	163934.4	8880995	11385891	12123916
b10	t1000	138121.5	15847861	13517167	13125185
b100	t1	93808.63	2192502	2404702	2817727
b100	t10	166666.7	12254902	18488389	20361498
b100	t100	174216	15873016	26199958	19412349
b100	t1000	94786.73	16051364	26223318	4026883
b1000	t1	98522.17	7262164	8935033	8679124
b1000	t10	99700.9	15873016	22880677	16682793
b1000	t100	162601.6	16129032	30217871	4509888
b1000	t1000	97847.36	9191176	1.05E+08	5147820

측정 결과, 위와 같은 표를 얻을 수 있었다. 이것을 그래프로 표현하면 다음과 같다.



측정 결과, 100^2 이상의 Grid 데이터를 처리할 때, Block과 Thread의 개수가 늘어나면 처리 속도가 더 증가하는 경향을 보였다. 다만 세가지 특징적인 점이 관찰되었다.

첫번째는 블록이 한 개일 경우, 스레드의 개수를 늘려도 유의미한 처리 속도 개선이 이루어지지 않았다는 것이다. 이는 블록이 하나일 경우 오로지 하나의 SM(스트리밍 멀티프로세서) 만이 작업을 할당 받기 때문에 병렬 처리를 달성하지 못하기 때문인 것으로 보인다.

두번째는 1000^2 그리드의 데이터를 처리할 때, 1000개의 블록, 1000개의 스레드가 있으면 다른 경우과는 차별화되게 속도 개선이 많이 이루어졌다. 다른 경우는 3E+8 cell/sec 이상의 속도를 달성한 경우가 없었으나, 이 특별한 경우엔 12E+8 cell/sec의 처리 속도를 기록하였다. 이것은 데이터의 구조와 블록/스레드의 할당이 완전히 맞아 떨어졌기 때문에 발생한 시너지라고 추측된다. 이 특이한 케이스는 총 1000^2개의 스레드가 각각 하나의 cell 연산만을 고정해서 담당하게 되고, 블록은 하나의 가로열만 할당됐기 때문에 하나의 SM 유닛이 하나의 가로열을 연산하게 된다. 이런 자료구조상의 처리 효율로 인해 cell/sec이 크게 증가한 것으로 보인다.

세번째는 그렇다고 해서 무조건 블록과 스레드의 크기를 높게 잡는 것이 좋은 성능을 보장해 주지는 않았다는 것이다. 오히려 100^2 과 10000^2의 그리드 크기의 경우엔 1000 블록, 1000 스레드의 프로세스가 최고의 속도를 보장하지 못했다.

위 세가지 경우를 살펴보면, 블록과 스레드는 고정된 값이 유리한 것이 아니라 GPU 구조의 가용 스트리밍 멀티프로세서와 스트리밍 프로세서 형태, 그리고 입력된 데이터의 환경에 따라 다른 처리 능력을 보여준다는 것을 알 수 있다.

4. 마무리 및 고찰

이번 과제는 CPU와 GPU의 병렬처리 속도를 직접적으로 비교해 볼 수 있었다. CPU 환경에서 작성된 프로세스는 콘웨이의 생명게임의 한 라운드를 진행하는데 상대적으로 많은 시간이 소모되었다. 특히 pThread를 이용한 CPU에서의 병렬 처리가 아닌, 단순 순차적인 프로세스는 데이터의 크기가 커진 경우 몇 십분 단위의 실행 시간이 걸리기도 했다. 하지만, GPU 환경에서 병렬 처리를 한 경우, 한번의 라운드에 매우 적은 시간이 필요했으며 CPU로는 몇 십분이 걸린 작업도 단 5초만에 끝냈다.

이것을 통해 병렬 처리를 위한 GPU 아키텍처 사용 전략이 매우 효과적인 것을 확인할 수 있었다. 물론 병렬 처리도 근본적인 알고리즘이 바뀌는 것은 아니고 단순히 병렬 처리 부분의 실행 속도를 1/n으로 낮추는 것이므로 데이터의 크기가 너무 커진다면 한계에 봉착할 것이다. 그래도 실용적인 목적에선 충분히 처리속도 개선의 가치가 있다고 볼 수 있다.

이런 GPU가 현실에서 활용되는 예는 컴퓨터 그래픽스, 그리고 암호화폐 채굴을 위한 연산기로의 사용 등이 있다. 컴퓨터 게임에서도 GPU는 필수적인 부품으로, GPU가 장착되지 않은 컴퓨터는 3D 그래픽 모델을 사용자와 실시간 상호작용하는 속도로 빠르게 만들어낼 수가 없다. 이번 과제를 통해 이러한 문제가 왜 발생하는지를 수치적인 검증을 통해 확인했다.

이번 과제를 진행하면서, 컴퓨터 구조의 형태는 어플리케이션 밑의 레이어에서 작동하면서, 그 어플리케이션의 성능에 지대한 영향을 미친다는 것을 알았다. GPU는 병렬 연산에 특화된 프로세서이기 때문에, 병렬 처리를 더 빠르게 할 수 있는 것처럼 다른 특수한 어플리케이션의 요구사항에 맞추어 하드웨어가 맞춤 제작된다면 어플리케이션은 더 고성능을 보장할 수 있을 것이다.

원래 물리적 컴퓨터 구조를 만드는 데에 드는 비용으로 인해 잘 시도되지 않았던 특수 구조 아키텍처가 최근 프로세서 성능 발전의 침체로 인해 산업 현장에서 도입을 시도하고 있다고 강의 시간에 들었다. 각 거대 it 기업들이 성능 향상을 위해 단순한 GPU 이용을 넘어 독자적인 컴퓨터 구조를 구축한다면, 미래엔 다시 기업별 특수 아키텍처에 따라 제공되는 다른 API를 개별로 배워야 기초적인 프로그래밍이 가능한 시대가 올 수도 있을 것 같다.