GPU요리하기_하드웨어전문가의 CUDA 개발자 가이드

목표:
GPU라는 주방의
재료(메모리)와
도구(SM, CUDA 코어)를
파악하고,
요리(프로그래밍)를 위한
기본 지식과 사고방식을
배운다

자료제공: NVIDIA (Jetson)

1.1 GPU 주방에 오신 것을 환영합니다

- 강사소개

  1985~2020: 어셈블리어, 8bit 마이크로 컴퓨터, 전자회로, 임베디드 시스템

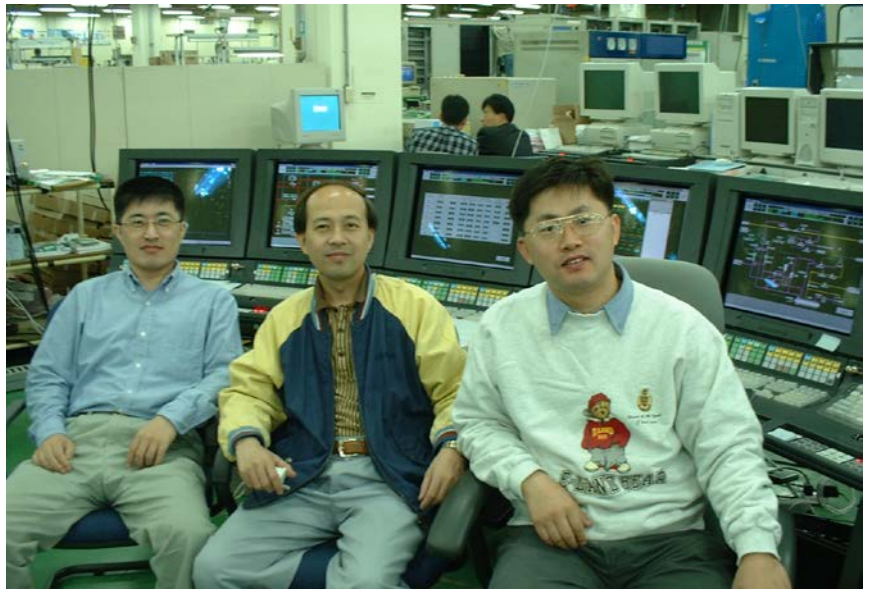
  1990~2010: 정유/석유화학 자동제어 (DCS,PLC)

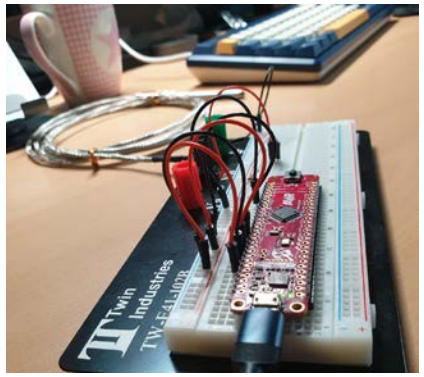
  전주대학교 차랑제어공학 겸임교수
  hysoft33@gmail.com

- 강의목적

  고수준 GPU 라이브러리 아래 깔린 하드웨어 동작원리, 프로그램에 미치는 영향

  고수의 개발자가 되는 길





1.2 GPU 주방에 오신 것을 환영합니다

- CPU: 지구인 일류 주방장
  - 못하는 요리가 없음 (금융처리, 게임기, 미사일 제어,..)
  - 순차처리 (Sequential Processing) 요리의 귀재

- GPU: 안드로메다 주방장
  - 수많은 재료를 동시에 처리해야만 하는 특별한 상황이 발생
  - 다니엘 힐 (1985년) 병렬처리 개발 정착

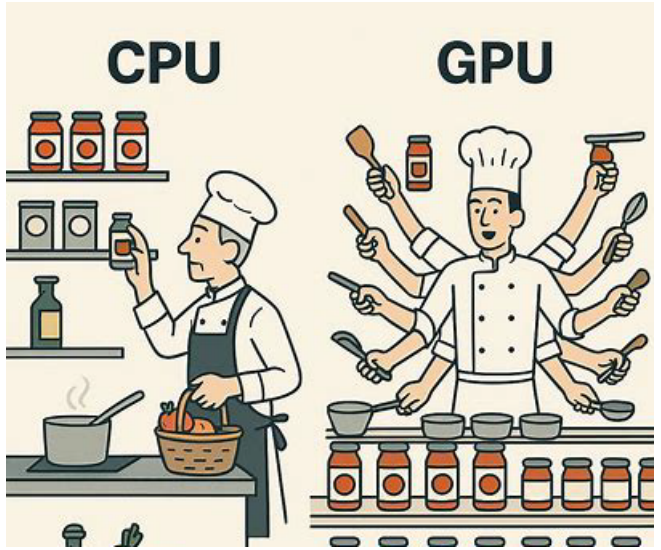
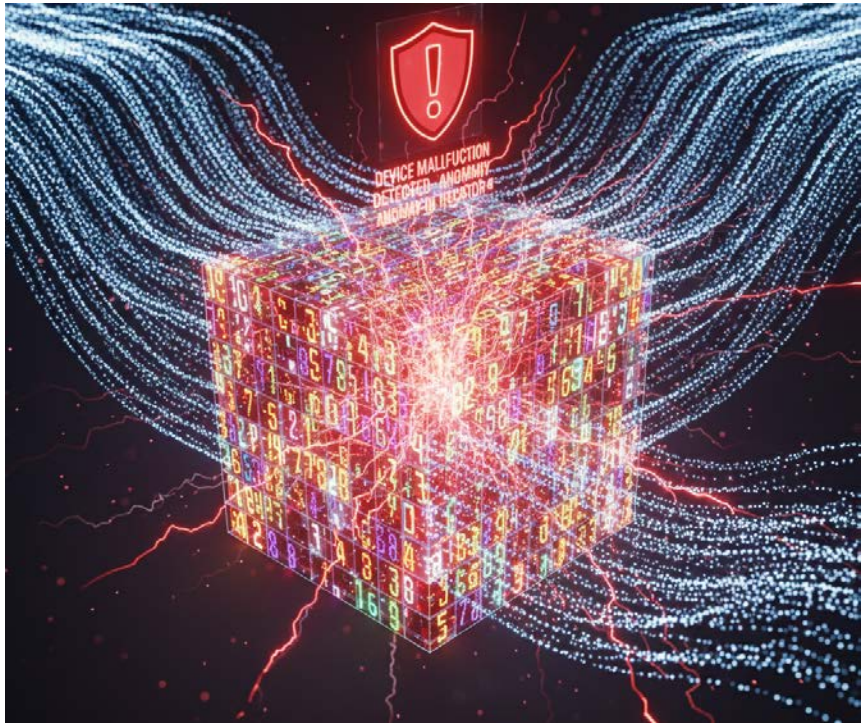


사진:Carniolus/CC BY 2.0

1.3 병렬요리 재료: 행렬

일상에서 만나는 행렬

1. 2차원 정적 행렬
   - 사과사진
   - 사과 사진의 한 픽셀 정보를 3개의 행렬 (R,G,B)로 표현
2. 3차원 동적 행렬
   - 2차원 데이터 + 시간흐름
   - 행렬 데이터가 동적으로 변함
   - 알고리즘이 센서값들의 변화추이를 고장 예지 경보

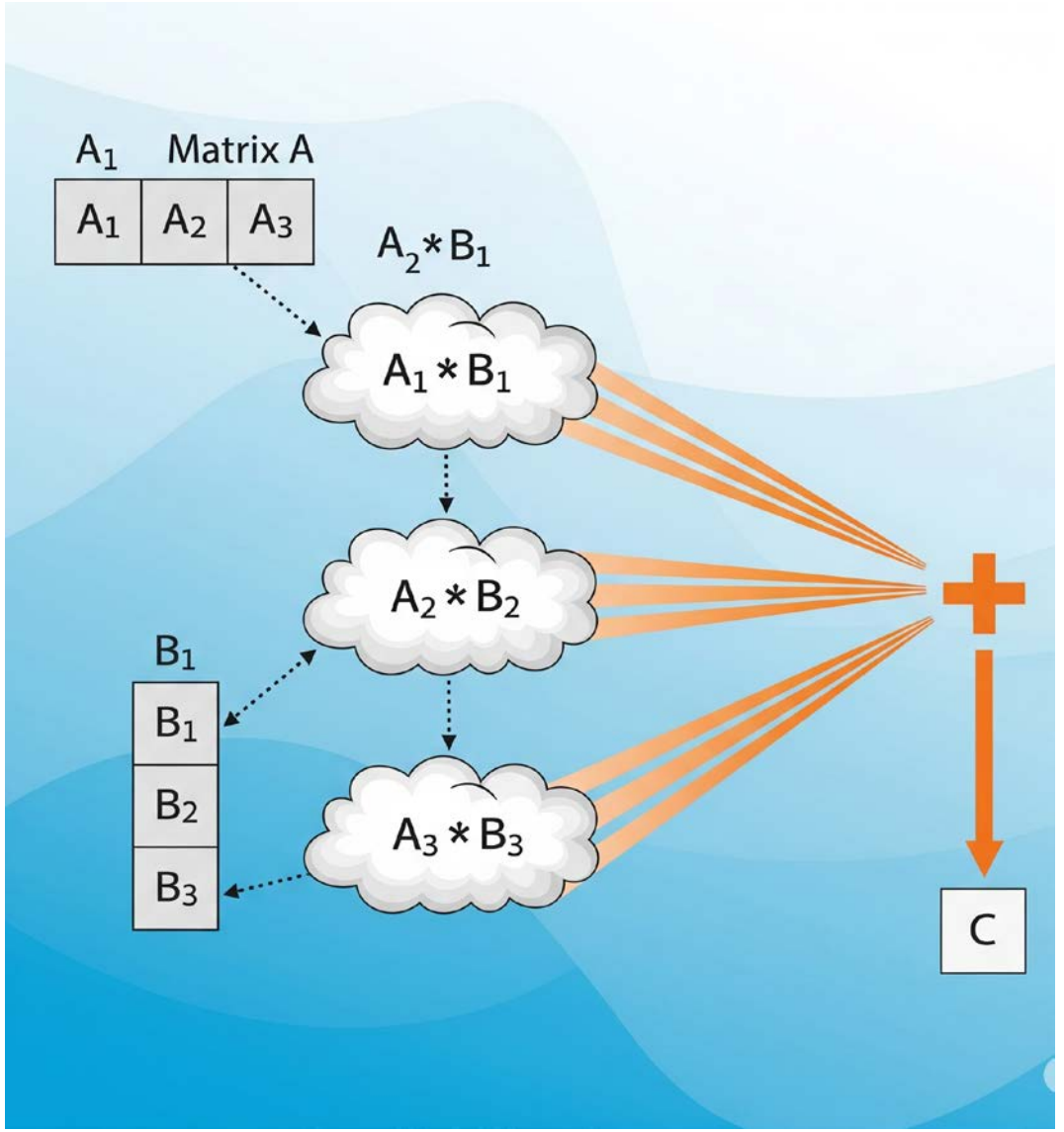


1.4 병렬요리 기본도구: 스레드

- 구름: 스레드 처리기
  - 정밀 사칙연산 계산기 (+,-,*,/), CUDA코어
  - 저정밀 사칙연산 계산기, Tensor 코어
  - 특수 계산기 (비교,논리,함수등), SFU
- 모든 스레드 처리기는 동시 시작 (병렬처리)
  - 그림: 처리기 3대가 동시 실행 상황
- 3개 스레드가 계산결과를 C에 반영 (동시에?)

  이 문제는 다음에..
- 그림은 C값 하나를 계산하는 경우임. 만일 A,B가 3*3행렬이라면, C값 또한 3*3행렬(원소9개)이므로 구름모양 스레드 9개가 동시 실행된다. (행렬의 곱)

(여담:사실 지구인에겐 이 개념이 어려워요, 안드로메다에선 자연스러울찌라도..)



1.5 병렬요리 기본도구: 워프

- 스레드 처리기의 구조(하드웨어)
  - CUDA코어 32 Set

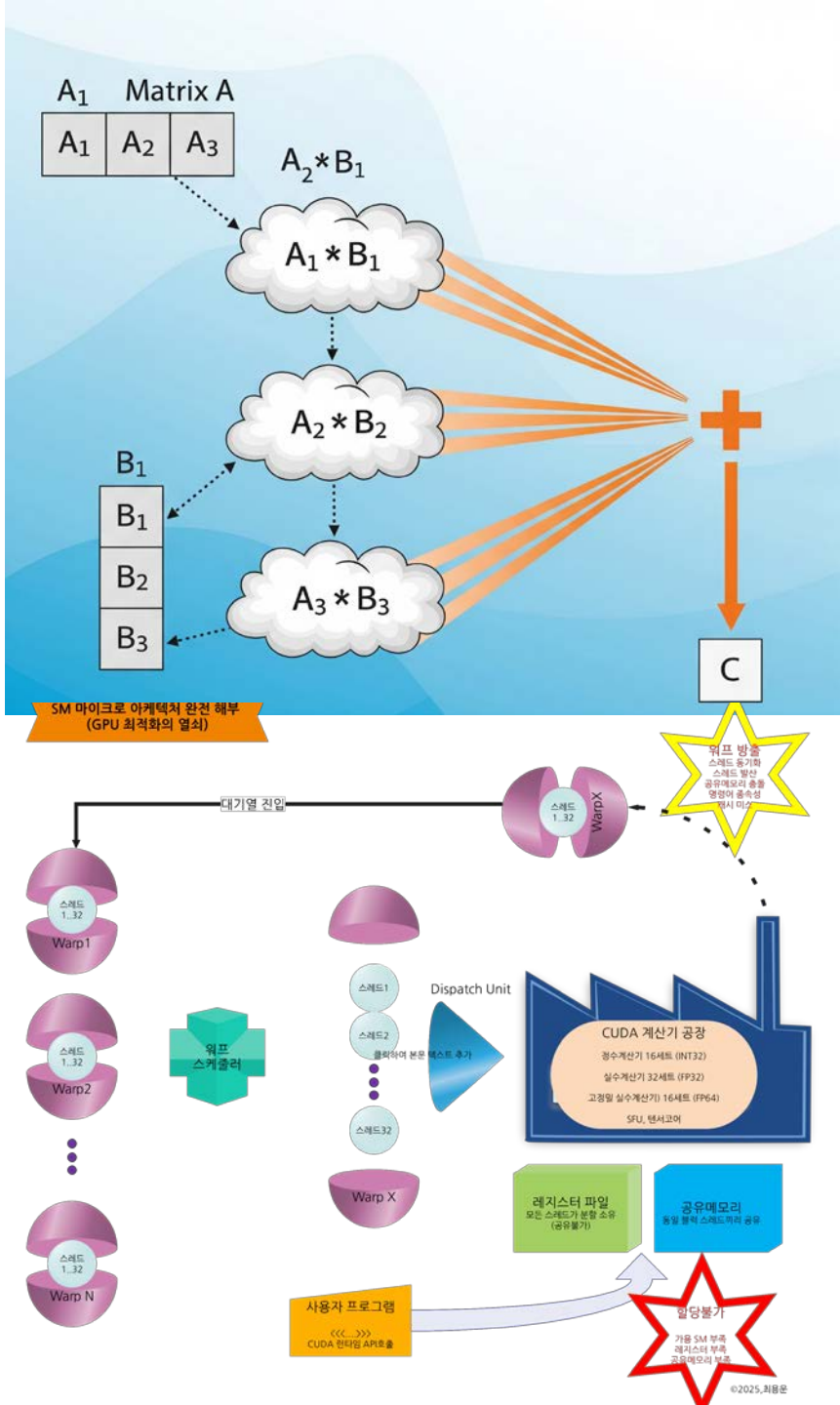
    INT32: 정수용 사칙연산기 16개

    FP32: 실수용(소숫점포함) 사칙연산기 32개

    FP64: 실수용(고정밀) 사칙연산기 16개
  - Tensor 코어
  - 특수 계산기 (비교,논리,함수등), SFU
- 스레드 처리기의 실행(하드웨어)

  워프 스케쥴러는 구름모양 32개 스레드의 실행을 주관하며, 1GHz속도로(초당 10억회) 하부에 있는 계산기들에게 작업을 배정,지시,관리한다.
- GPU칩 설계시 32개 스레드 처리로 고정

  동시 처리되는32개 스레드 묶음을 워프(Warp)라고 함



1.5 병렬요리 기본도구: SM(일반)

- SM: Streaming Multiprocessor

  Streaming: 데이터가 마치 물줄기 처럼 끊임없이 올러 들어오는, 그래픽 데이터나 딥러닝 데이터처럼 대규모 데이터 흐름을 처리하는데 최적화된 특성을 의미

  Multiprocessor: 한 개의 연산 프로세서가 아니라 여러 개의 코어(쿠다,텐서)와 스케쥴링 및 자원관리기능까지 갖춘 완전한 프로세서 묶음

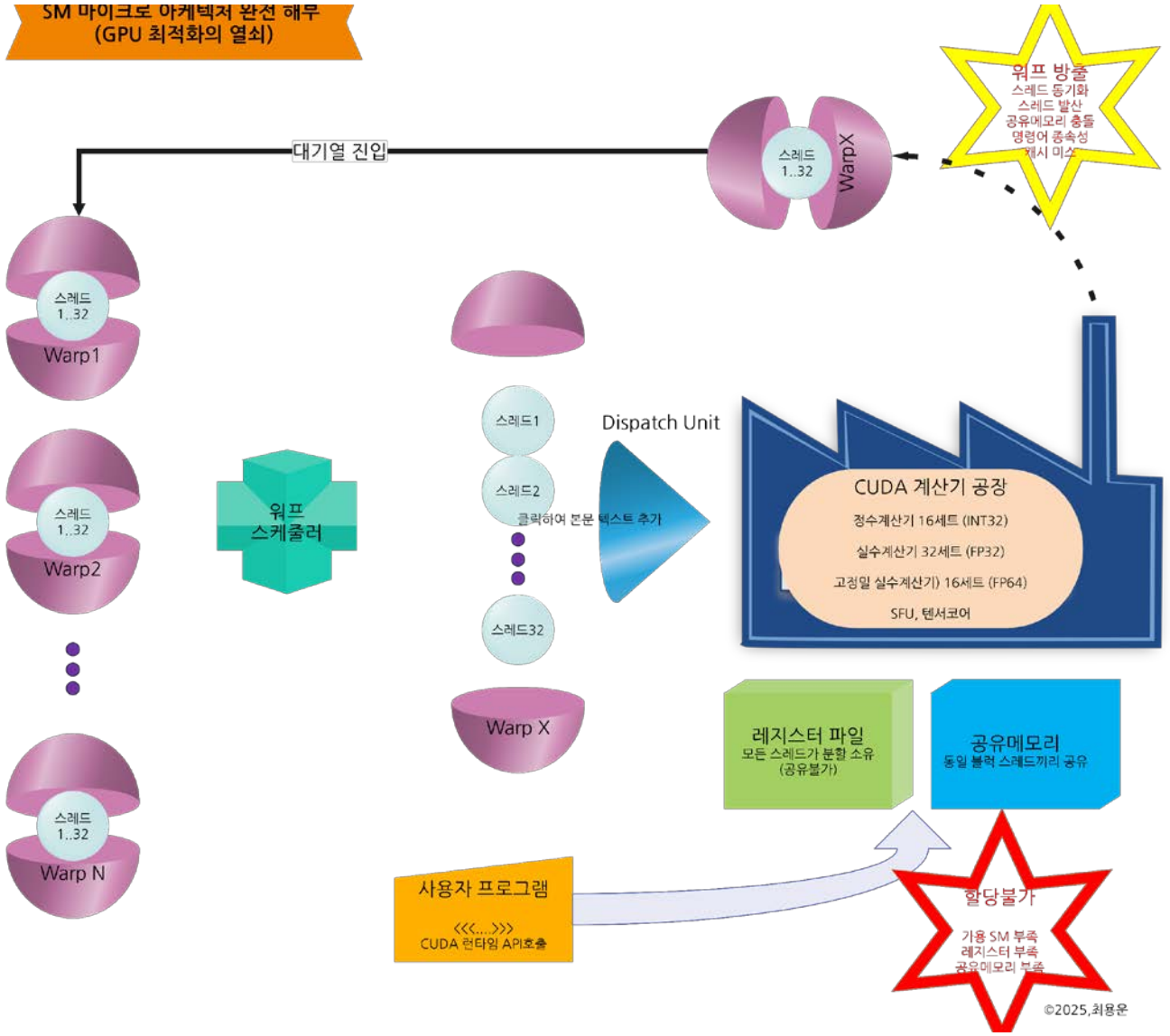
- SM 태동배경

  2000년 중반, GPU의 많은 코어로 행렬등 과학계산 시도 시작

  2006: NVIDIA, 누구나 쉽게 병렬프로그램밍 가능하도록 CUDA발표, SM구조에서 실행

- 병렬처리 추상화

  개발자 입장에서 SM은 블랙박스 임. 개발자가 그리드와 스레드 블록을 정의하여 작업을 던져주면, SM은 알아서 가장 효율적인 방식으로 자원을 할당, 스케쥴링하여 작업을 처리한다.



1.5 병렬요리 기본도구: SM (하드웨어)

- SM: Streaming Multiprocessor

  NVIDIA 창업 젠슨황 및 동료들, ASIC설계분야 베테랑,단순 그래픽카드 탈출,목적에 따라 프로그래밍이 가능한 연산용 ASIC칩 제작목표,SM 탄생 (참고 ASIC: 특정목적에 따라 설계된 직접회로)

- SM은 NVIDIA의 가장 중요한 핵심기술,다양한 기능을 수행하는 ASIC 블럭들,하나의 칩에 통합된 디지털 전자회로

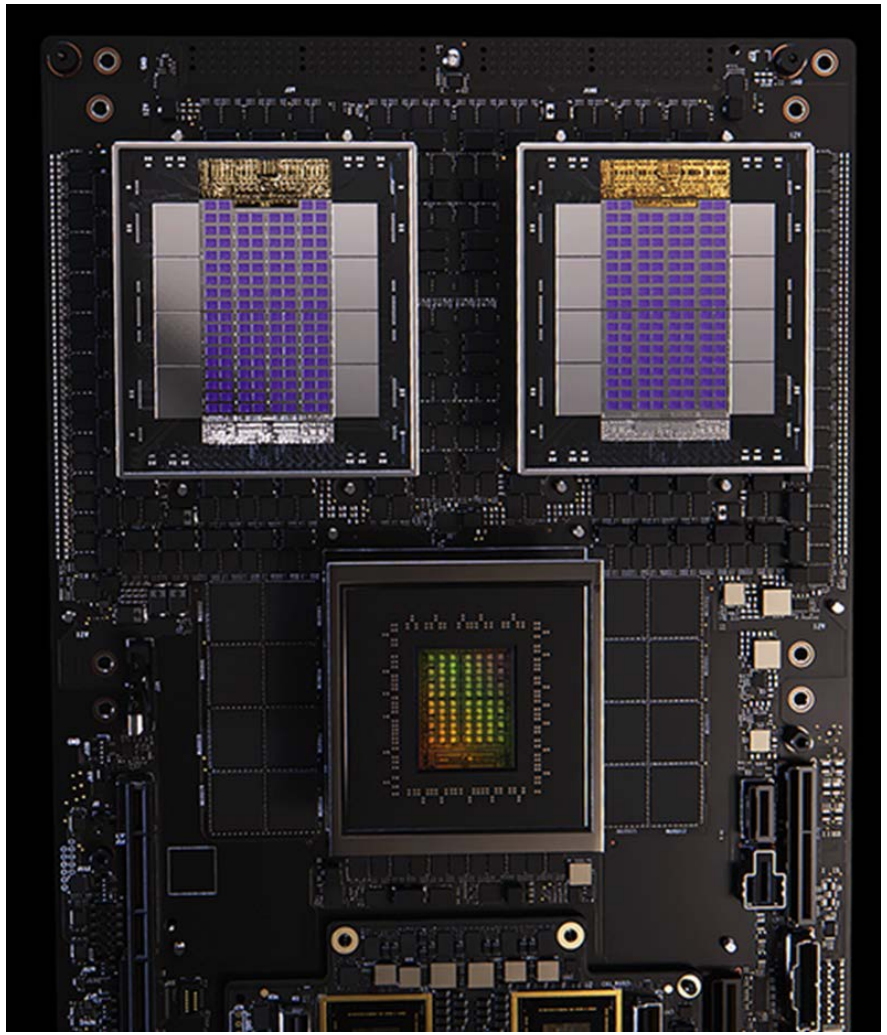
  워프 스케쥴러, 디스패처: 32개 스레드의 실행을 분배, 지시

  쿠다코어,텐서코어: 프로그램 코드에 포함된 계산기능 처리

  SFU: 판단, 분기, 수학함수 계산 처리

- GPU종류에 따라 내장된 SM 개수가 다름

  보급형: 10개 내외, 고급형:50~90개, 센터용:100개 이상



(자료제공:NVIDIA)

1.6 병렬요리 기본도구: 블록의 크기

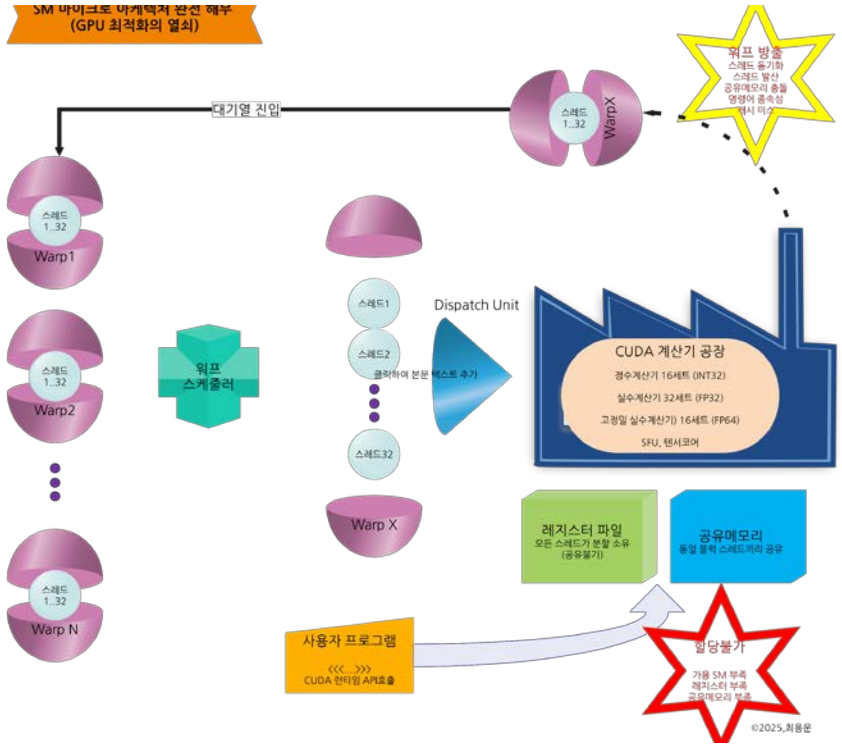
- 스레드 블록 크기는 32의 배수

  원칙: 블럭크기는 하드웨어 사양에 적합해야 한다

  NVIDIA GPU의 경우 워프 크기(32)의 배수 128,256,512,1024등으로 한다

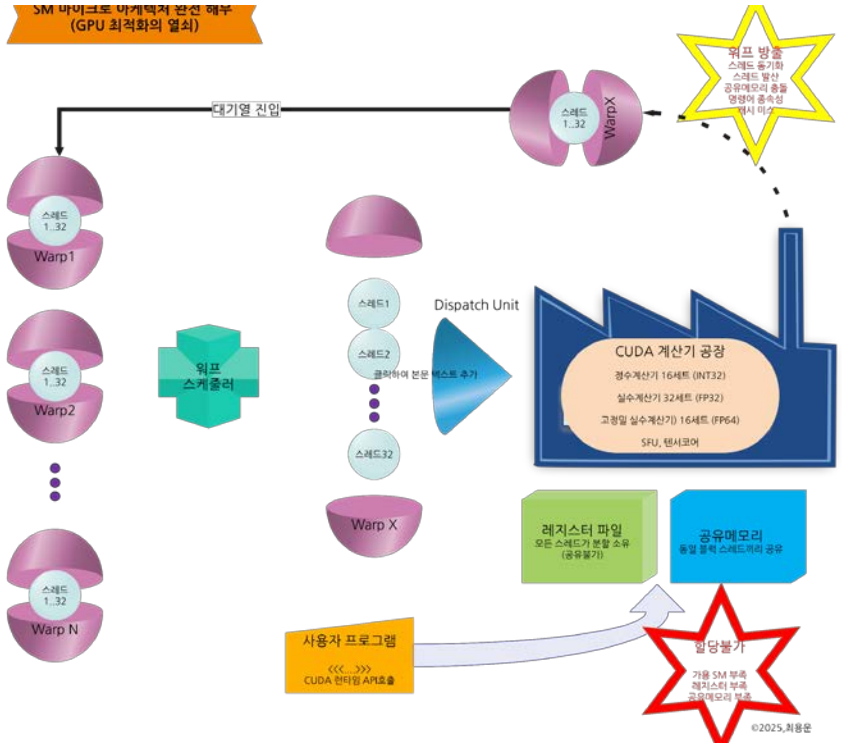
- 큰 블록 (512,1024) 장점 (워프 스케쥴러 안에 일거리가 가득 채워짐)
  - 높은점유율, 지연숨김: 실행 중인 스레드에서 지연상황(메모리 대기) 발생하면, 스케쥴러가 즉시 다른 스레드 실행으로 교체하기에, SM 유휴시간이 줄어든다
  - 효율적 스레드 간 통신:동일 SM에 할당된 모든 스레드는 SM의 공유메모리(SRAM,128~256kB)를 사용하여 데이터 공유하기에, 느린 전역메모리(HBM)사용 방법보다 높은 성능을 제공한다.
- 큰 블록 (512,1024) 단점 (스레드 마다 기본자원이 할당되기에, 자원부족 발생)
  - SM의 자원인 레지스터 메모리를 각 스레드 마다 전용으로 할당해 주기 때문에, 레지스터 메모리 부족으로, 동시실행 스레드 개수가 줄어든다.
  - 스레드 발산 관리 어려움: 스레드별로 if-else 조건에 따라 다른 코드를 실행하는 현상, 이 상황에서 스레드 수 까지 많으면, 동기화 어렵고, 실행이 길어진다

  참고 SM의 if-else의 처리방법: 예를 들어 32개 스레들 중 if조건이 참인 스레드 16개, 거짓 조건인 스레드 16개라면, 일단 거짓조건인 스레드 모두 비활성 처리하고 참인 스레드의 코드 실행을 지속하여 마치고, 이번엔 반대로 거짓조건인 스래드 실행을 활성화시켜서 코드실행을 지속한다. (즉 실행기간이 2배 길어진다)

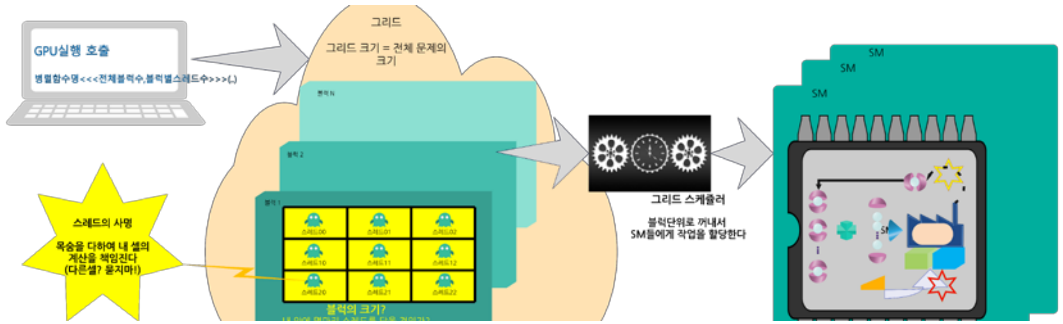


1.6 병렬요리 기본도구: 블록의 크기

- 적은 블록 (64,128) 장점 (워프 스케쥴러에 여유발생, 다른 블록과 동시작업)
  - 낮은 자원 소모: SM의 공유자원 소모가 적기 때문에, 워프 스케쥴러가 또 다른 블록의 동시 할당하여 전반적으로 GPU 점유율을 높일 수 있다.
  - 유연한 스케줄링: 작은 블록들이 많아지면, SM의 어프 스케쥴러가 작업량이 끝난 블록을 바로 다른 블록으로 교체하는 등 스케쥬링의 유연성 높아진다.
  - 스레드 수가 적기 때문에, 조건문등으로 인한 스레드 발산 영향이 상대적으로 적다
- 적은 블록 (64,128) 단점
  - 단일블록 만으로는 SM의 모든 실행유닛에게 충분한 일거리를 주지못함으로 일부 코어는 놀게된다.
  - 스레드 간 통신 비효율: 만약 스레드간 데이터공유가 필요할 경우, 블록크기가 작기때문에 이웃한 스레드(이웃한 데이터 처리를 담당하는 스레드)가 다른 블록에 소속되어, 느린 HBM을 통한 공유를 사용하기에 성능 저하가 발생한다.
- 결론: 128~1024 사이에서 여러 크기를 실험하여 가장 성능이 높은 블록크기로 결정한다. (결론은 단순한데, 왜 어려운 설명을 했을까요?)
  - 장단점을 잘 알고 있는 사람은 성능문제 발생시 정확한 진단과, 바른 개선 방향성을 찾는다.
  - 장단점을 알고있는 사람은 문제가 발생하기 전에 예측하여 미리 방지할 수 있다.



1.7 병렬요리 기본도구: 그리드와 데이터



- 1.데이터 중심적으로 설계된 GPU

  GPU는 방대한 데이터를 동시처리하는 구조이며, 미리 배치된 수많은 처리기(쿠다코어)에 데이터를 효과적으로 공급하는 것이 GPU 설계의 핵심이다

- 2.그리드와 데이터 관계

  그리드는 GPU가 처리해야 할 전체 데이터의 논리적 배열

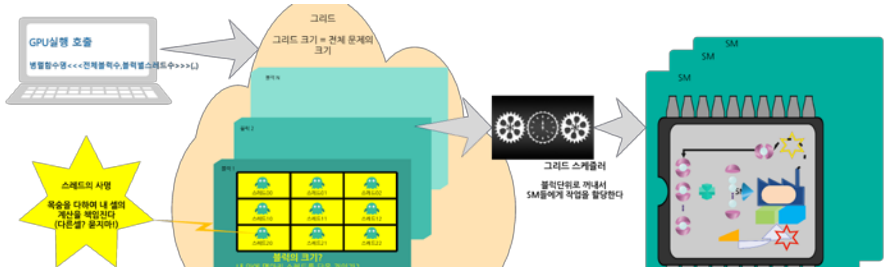
  GPU 아키텍처는 근본적으로 데이터 중심적(Data-centric) 철학을 바탕으로 하였음

  예: 행렬 A•B = C

  GPU는 결과값 행렬C의 하나의 원소를 계산하는 것을 '하나의 작업'으로 간주, 모든 작업을 동시처리하려 하기에, 총 작업 수량 = 행*열

  즉, 그리드 크기는 하드웨어적 제약과 별개로 프로그래머가 처리해야 할 데이터의 크기와 모양에 맞춰 정의해야 한다.

1.7 병렬요리 기본도구: 그리드와 데이터



- 3.스레드 블록: 하드웨어와 연결되는 관문

  스레드 블록은 그리드를 나누는 작은 논리적 덩어리

  기본적으로 그리드 스케쥴러는 여러 SM들에게 스레드 블록을 균등 배분한다 (SM의 유휴 방지, GPU 점유율 상승)

  동일 블록이 다수의 SM에게 나뉘어 배정 불가

  규모가 작은 다수의 블록은 동일 SM에게 배정될 수 있다.

  동일 블록안의 스레드들은 SM 공유 메모리를 공유한다 (단,스레드 자신의 레지스터는 절대 공유 불가)

  스레드 블록 크기 결정은 SM의 구조와 물리적 제한을 우선 고려한다

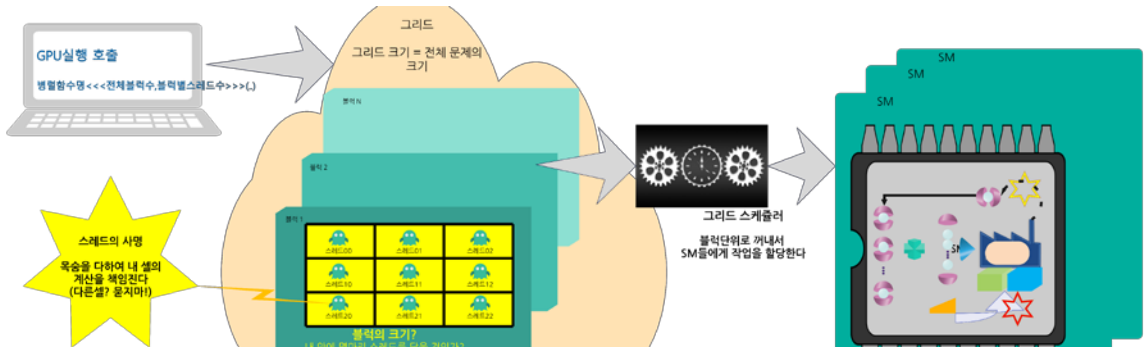
- 4.워프와 스레드

  스레드는 SM에서 실행되며, 하나의 데이터 요소에 대한 연산 담당 (데이터 중심적 철학)

  워프는 SM이 관리하는 32개 스레드 묶음, 동일 워프는 동일명령을 동시 실행한다

  동일 워프내 스레드의 실행경로가 갈라지는 현상(스레드 발산) 발생하면 성능저하 발생한다.

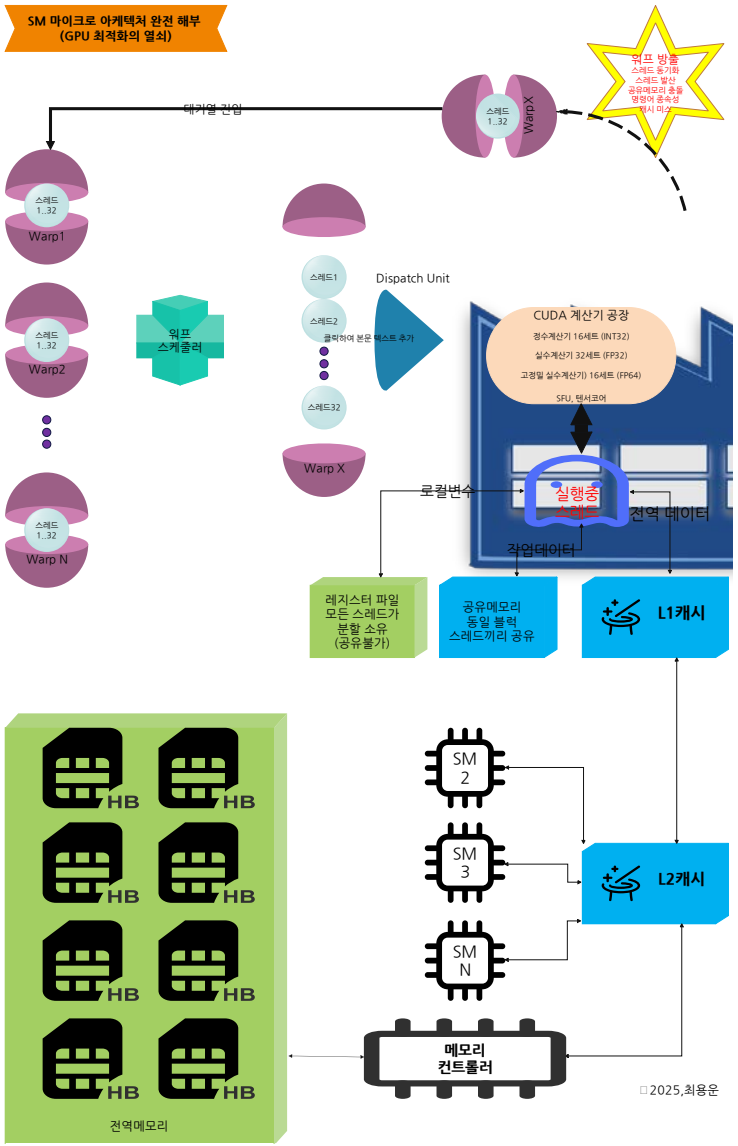
1.8 병렬요리 기본도구: 작업과정 요약



- 1. 데이터 준비: 처리할 데이터의 크기 파악 (최종 결과값을 담는 행렬의 크기가 기준)
- 2. 그리드 결정: 데이터 크기따라 그리드 차원 설정
- 3. 블록 분할: 하드웨어 자원 효율성 고려하여 블록크기 결정
- 4. SM할당: GPU에 내장된 글로벌 스케쥴러가 자동으로 수행
- 5. 워프실행: SM에 내장된 워프 스케줄러가 워프단위(32개 스레드)로 실행지시, 관리

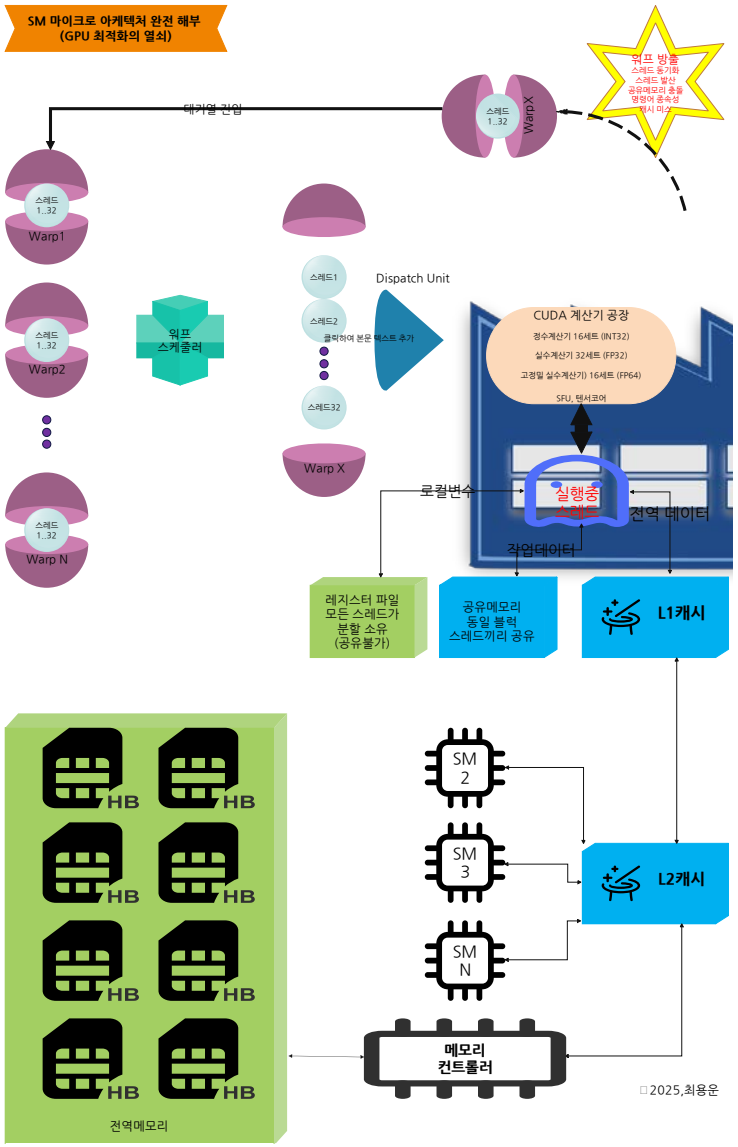
1.9 병렬요리 기본도구: 메모리창고

- 1.개요: 왜 메모리 계층구조를 알아야 하나?
  - GPU의 연산속도가 메모리 접근 속도보다 1000배 이상 빠르기 때문에,
  - 메모리 계층구조를 정확히 알면, GPU의 엄청난 병렬 연산 능력을 100% 활용할 수 있다.
- 2.개발자가 GPU속도를 획기적으로 높이는 방법들
  - 스레드 내부에서 사용하는 변수는 무조건 로컬변수를 사용하되 변수 개수는 최대한 절제하여 사용한다
  - 동일 블록에 속한 여러 스레드가 반복적으로 사용하는 데이터는 공유(__shared__)메모리로 변경한다
  - 행열계산은 행단위(행번호 고정, 열번호 바꿔가며 계산) 계산방식으로 작성하되, 작업단위는 32의 배수 관계로 작성한다. (배열은 행 순서대로 전역메모리에 저장되는 구조이며, 행렬 계산에 돌입한 32스레드(워프)들은 제작각 동일행에 열번호만 달리한 데이터를 요청할 것이며, 메모리 컨트롤러는 이런 요청을 하나로 묶어서 처리하기에 처리성능이 올라간다.



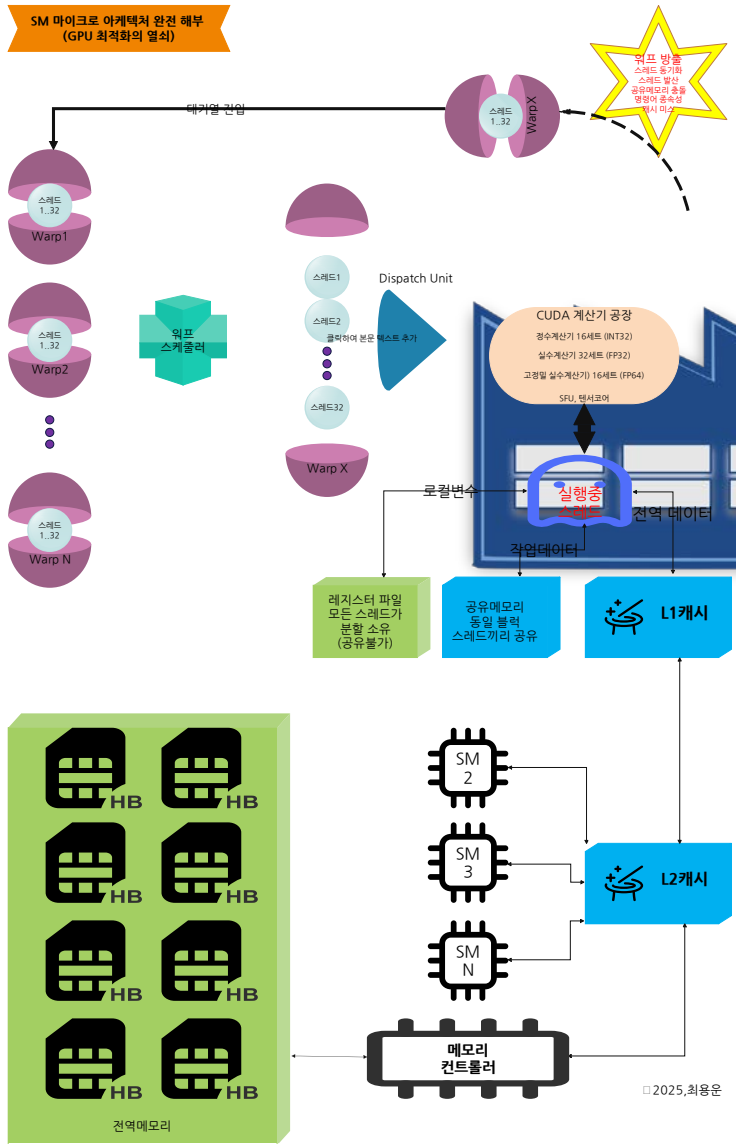
1.9 병렬요리 기본도구: 메모리창고

- 3.레지스터: 각 요리사(스레드)의 앞치마 주머니
  - GPU에서 가장 빠르고, 용량이 가장 작은 메모리 (SM 내부에 배치되어 있음)
  - 스레드 자신만 사용할 수 있음, 로컬변수 저장소 (다른 스레드 접근 불가)
- 4.공유 메모리: 동일 요리팀 (스레드블럭)의 조리대 위 선반에 위치
  - 동일 블록에 속한 스레드들끼리 공유
  - 같은 SM 내부에 있지만, 레지스터 보단 느림
  - 사용자가 느린 전역메모리(HBM) 내용 일부를 이곳에 옮겨두고 사용 (성능향상효과 높다)



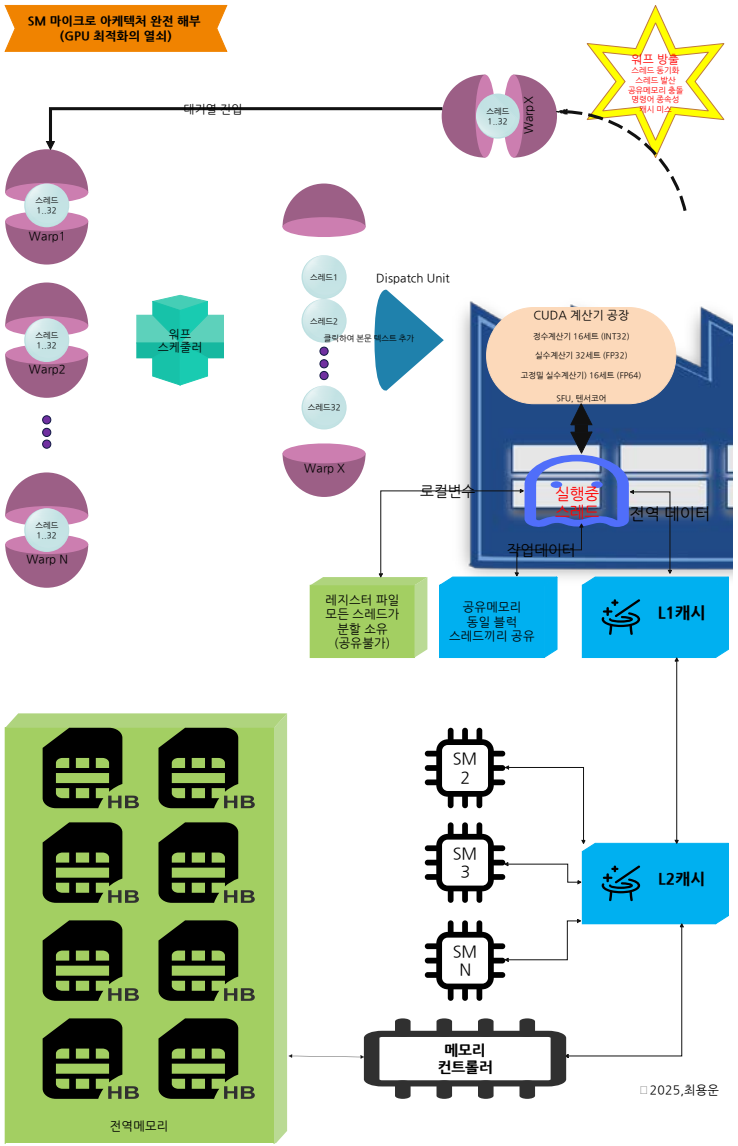
1.9 병렬요리 기본도구: 메모리창고

- 5. L1/L2 캐시 (Cache)
  - 자주 접근하는 데이터를 느린 HBM에서 가져와서 이곳에 임시 저장
  - 각 SM별로 전용 L1 캐시가 연결돼 있고, L2 캐시는 모든 SM에게 공통으로 연결돼있음
  - 하드웨어(메모리 컨트롤러가) 캐시관련 모든 일을 처리 (프로그래머는, 아 이런 구조가 있구나~)
- 6.전역메모리 (HBM)
  - 용량은 가장 크지만, 속도는 가장 느림, DRAM
  - (사실 HBM속도는 최신 DDR메모리보다 10배이상 빠르지만, GPU보다는 10배이상 느림)
  - GPU에서 실행중이 모든 스레드가 접근 가능
  - 데이터를 GPU에 전달하기까지 수백 사이클(스레드가 명령 수백번 실행 기간임) 지연 발생
  - 전역메모리 접근 최소화가 고수 프로그래머



1.9 병렬요리 기본도구: 메모리창고

- 5. L1/L2 캐시 (Cache), 마법사 상주
  - 자주 접근하는 데이터를 느린 HBM에서 가져와서 이곳에 임시 저장
  - 각 SM별로 전용 L1 캐시가 연결돼 있고, L2 캐시는 모든 SM에게 공통으로 연결돼있음
- 캐시에 상주하는 마법사
  - 지연시간 속이기: 스레드가 데이터를 요청하였는데, 해당 데이터가 캐시에 없음! 마법사가 워프에게 잠시멈춤 요청함에 따라, 즉시 다른 워프가 실행됨
  - 미래 예측:스레드들의 데이터 요청 패턴을 분석하고, 컴파일러 실행 코드를 분석하여 필요한 데이터를 미리 가져다 놓는다.
  - 공간 청소: 캐시 메모리가 가득차게되면, LRU(가장 오랫동안 사용되지 않은 데이터) 순으로 메모리에서 삭제한다.



1.9 병렬요리 기본도구: 메모리창고

- 6. 데이터를 찾아 떠나는 여행 by Thread
  - ① 스레드가 d_A[idx], d_B[idx] 실행순간 여행이 시작된다
  - ② 가장 가까운 레지스터 탐색 (빠르지만, 용량이 작아 없을거임)
  - ③ 공유 메모리 탐색(혹시 다른 스레드가 가져다놓았다면!)
  - ④ L1탐색 (L1에게도 없다면 L2 마멉사에게 요청)
  - ⑤ L2탐색 (L2에게도 없다면, 장거리 여행 시작, 메모리 컨트롤러에게 요청)
  - ⑥ 메모리컨트롤러는 여러 개의 작은 요청들을 결합하여 한 번의 대량읽기로 처리하고, 읽어온 자료를 분류하여 각 요청에 응답한다

C++

```
__global__ void add(int *d_C, int *d_A, int *d_B) {
    // 1. 스레드의 고유 번호(idx)를 계산합니다.
    //    이것은 스레드가 '자신이 처리할 데이터가 무엇인지'를 파악하는 과정입니다.
    int idx = blockIdx.x * blockDim.x + threadIdx.x;

    // 2. '데이터를 필요로 하는 실행문'입니다.
    //    이 한 줄이 바로 '여행의 시작점'입니다.
    //    스레드는 d_A[idx]와 d_B[idx]의 값을 찾아오라고 요청합니다.
    d_C[idx] = d_A[idx] + d_B[idx];
}
```