C++ *AMP(Accelerated Massive Parallelism)*

2013-03-29

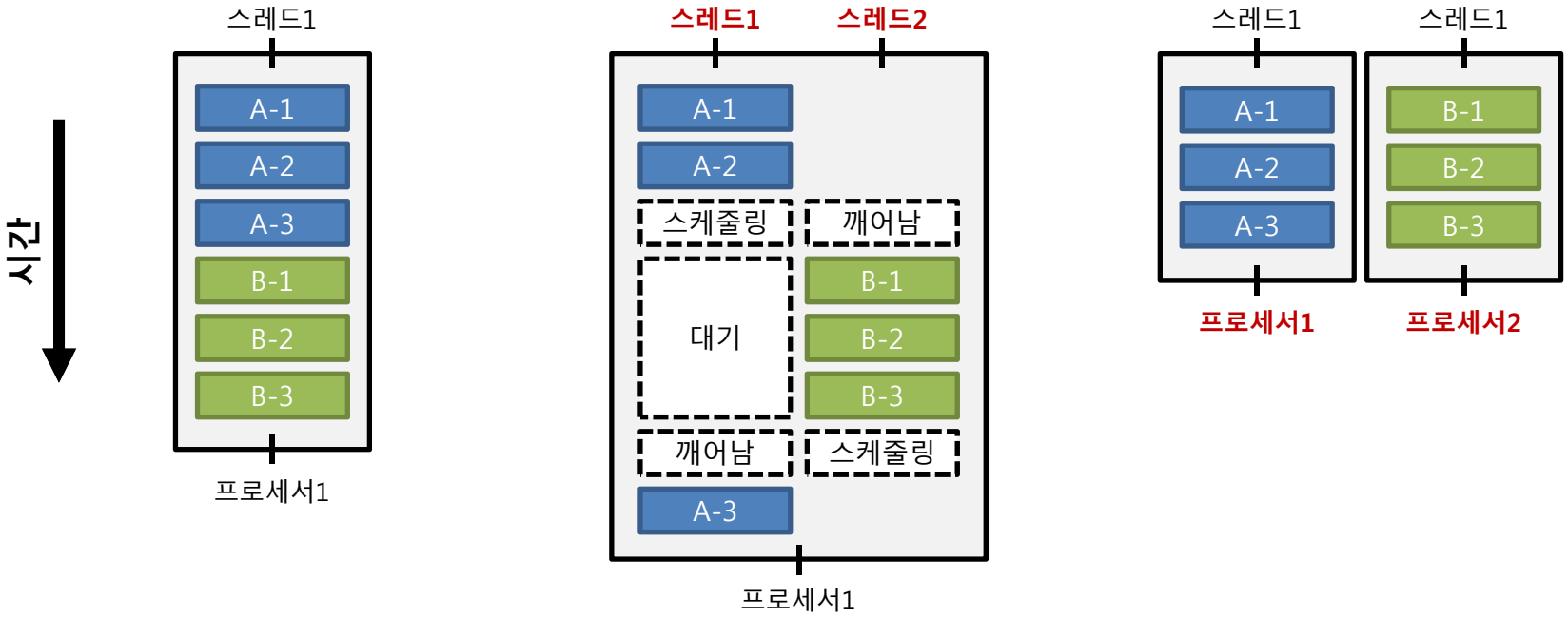
고형호
hyungho.ko@gmail.com

http://hhko.tistory.com

목차

Part 1. 다중 프로세서

직렬 *vs.* 병행성 *vs.* 병렬성



직렬
(Serial)

병행성
(Concurrency)

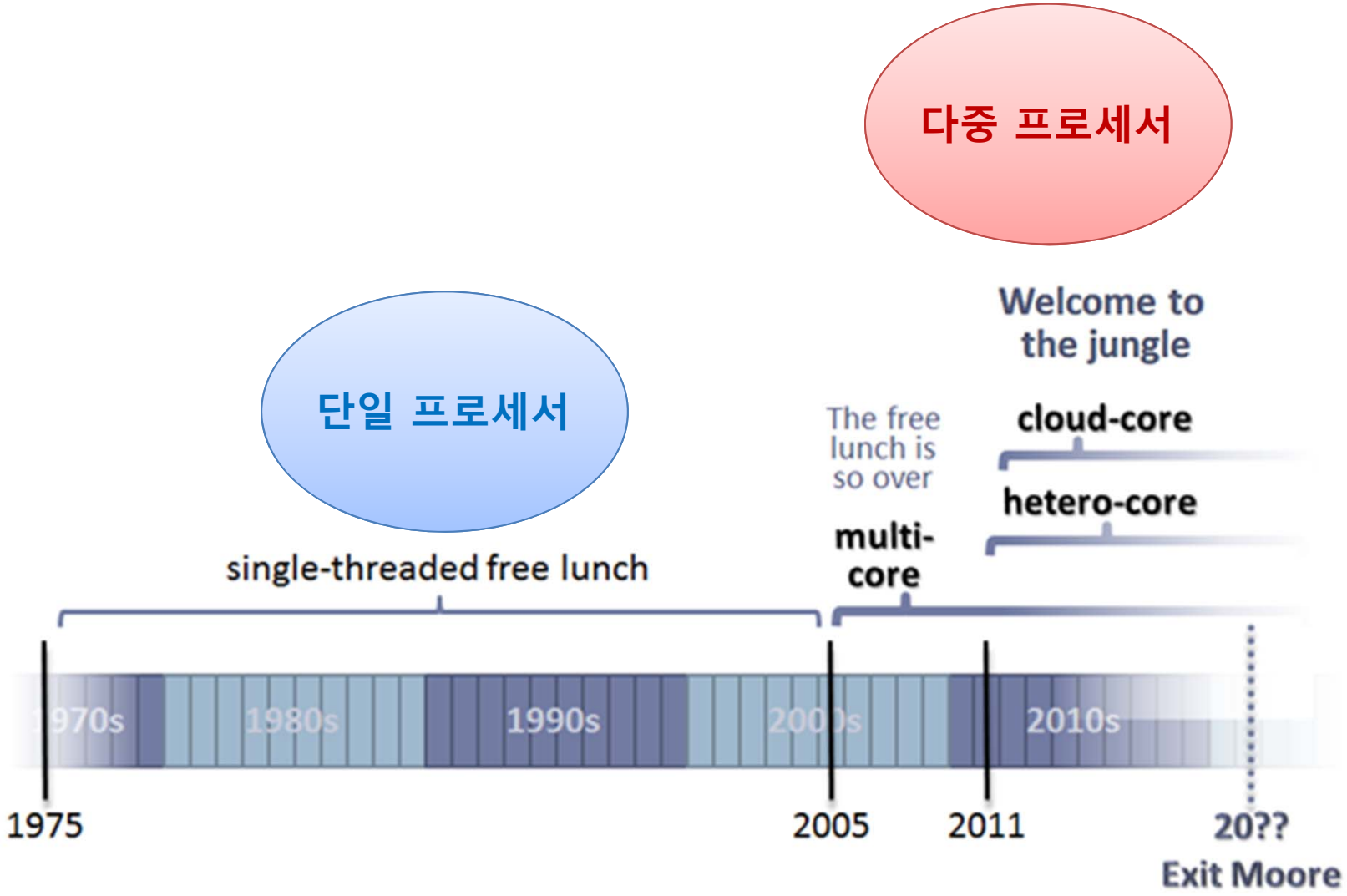
병렬성
(Parallelism)

단일 프로세서

단일 프로세서

다중 프로세서

프로세서 발전사


다중 프로세서
단일 프로세서
Welcome to the jungle
The free lunch is so over
cloud-core
hetero-core
multi-core
single-threaded free lunch
1970s
1980s
1990s
2000s
2010s
1975
2005
2011
20??
Exit Moore

Part 2. 연산 자원

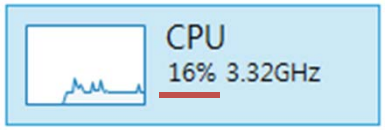
CPU

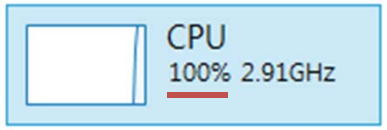
CPU(multi-core, 다중 프로세서) 자원 활용하기

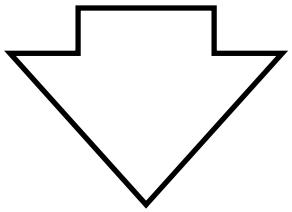
```
while(true)
{
}
```

직렬 *Vs.* 병렬

```
parallel_for( ...
{
});
```







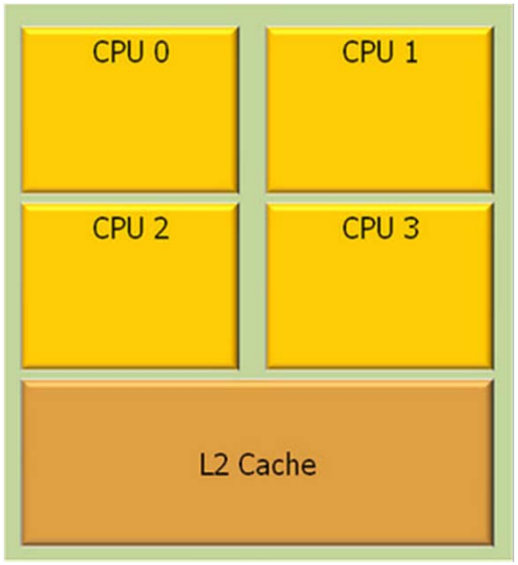
이제 CPU 자원 활용은 필수다.

GPU

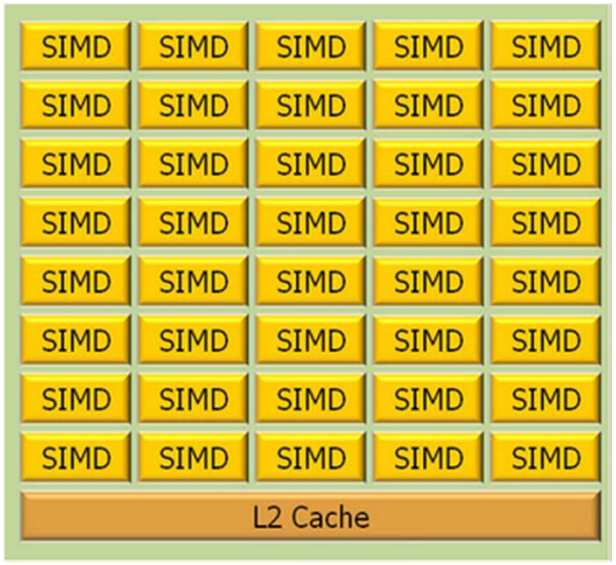
연산 능력을 갖고 있는 자원은 CPU뿐인가?

GPU(Graphics Processing Unit)

그래픽 카드에 장착된 그래픽 연산장치

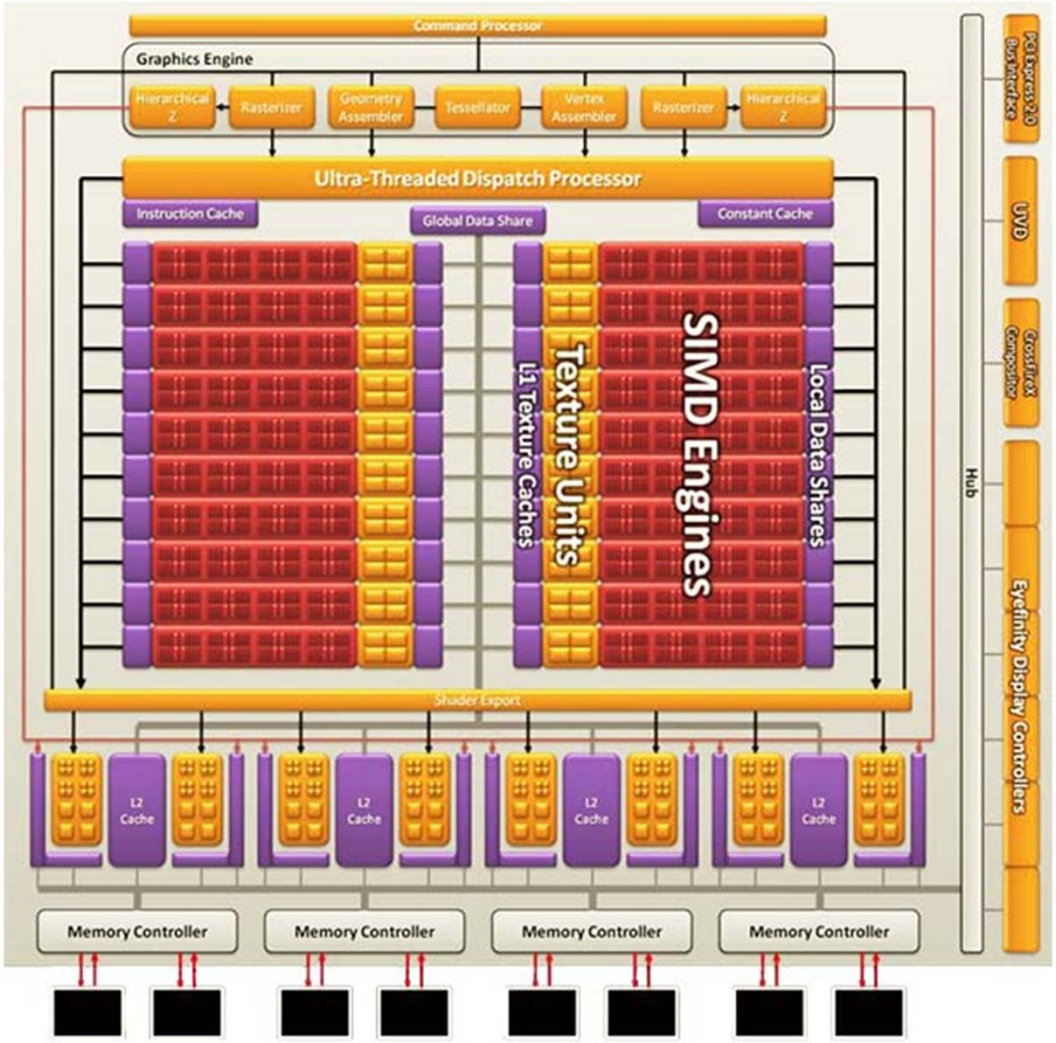


CPU



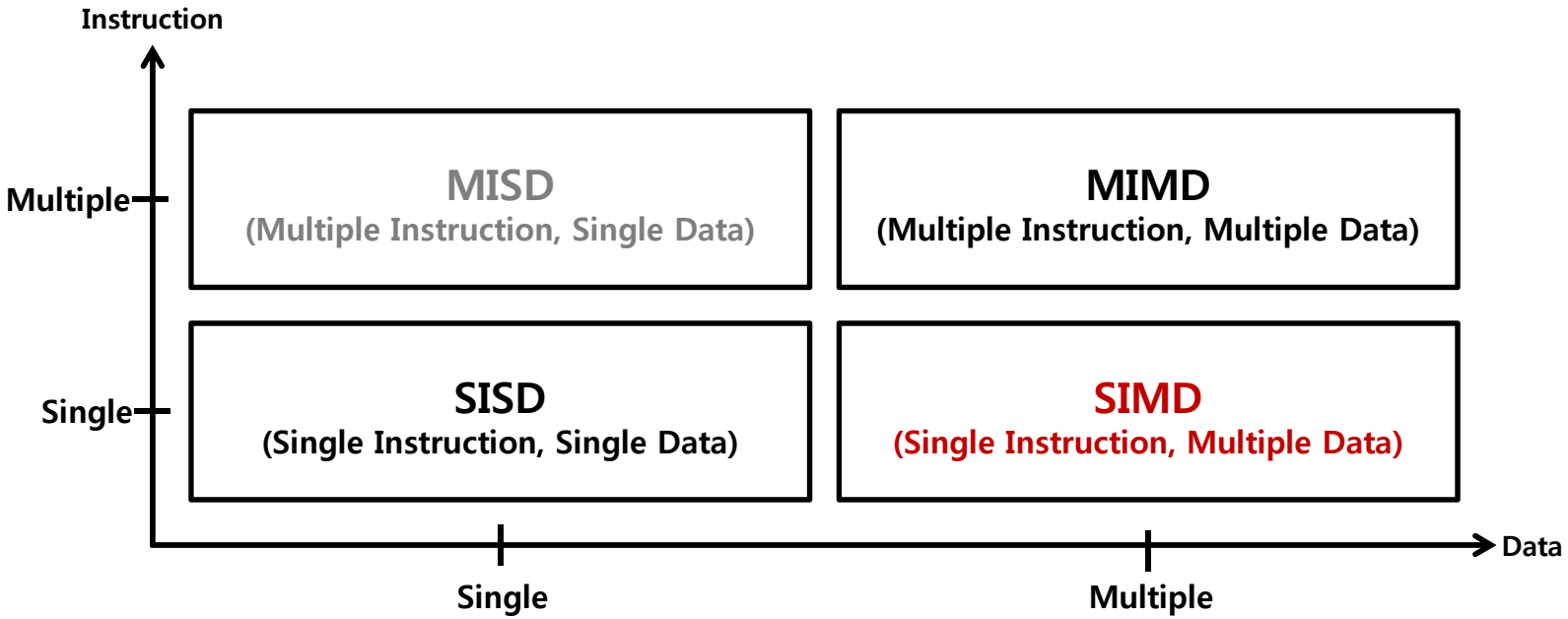
GPU

AMD ATI Radeon HD 5870

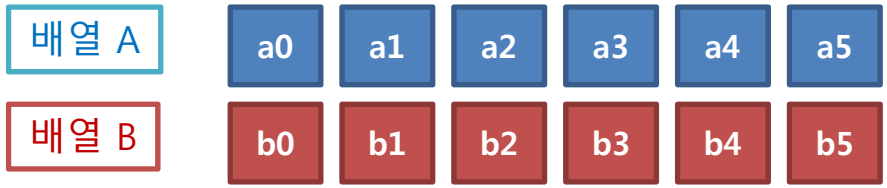


SIMD은 **GPU 연산의 핵심**이다.

병렬 하드웨어 아키텍처 분류

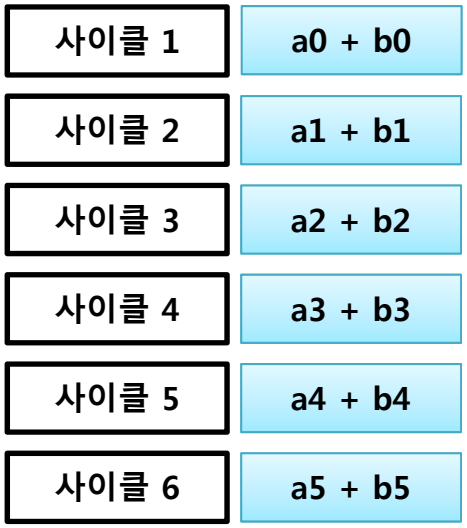


SISD *vs.* SIMD



배열 A와 B를 더하여라!

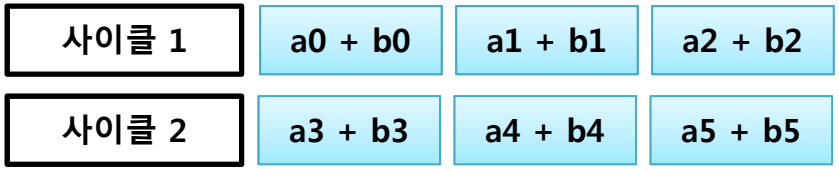
단일 명령으로 단일 데이터 처리



SISD

(Single Instruction stream, Single Data stream)

단일 명령으로 다중 데이터 처리



SIMD

(Single Instruction stream, Multiple Data stream)

SIMD 발전사

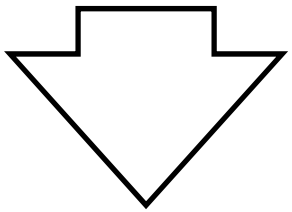
연도	CPU	CPU 확장 명령어 셋	벡터 처리 능력 (벡터 레지스터 수)
1996			
1997	펜티엄 P55C	MMX Multi Media eXtensions	64 비트
1998			
1999	펜티엄 3 카트마이	SSE Streaming SIMD Extensions	128 비트
2000			
2001	펜티엄 4 윌라멧	SSE 2	
2002			
2003			
2004	펜티엄 4 프레스캇	SSE 3	
2005			
2006			
2007	코어2 메롬		
2008	코어2 펜린	SSE 4	
2009	코어 i7 네할렘		
2010	웨스트미어		
2011	샌디 브릿지	AVX Advanced Vector Extensions	256 비트
2012			
2013			

SIMD 예제

```
for (unsigned int i = 0; i < MAX_SIZE2; i++)
{
        view_c[i] = view_a[i] + view_b[i];
}
```

view_c[i] = view_a[i] + view_b[i];

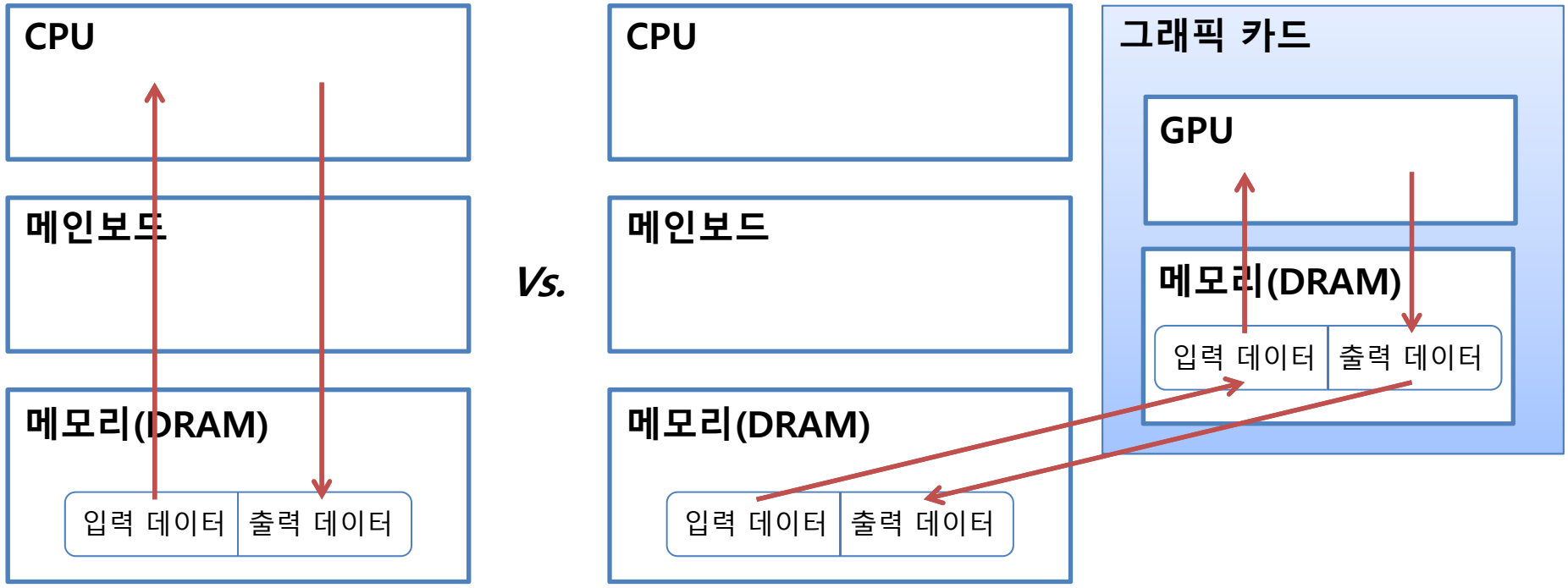
```
00AC121B   mov       eax,2625A0h
00AC1220   movdqu    xmm1,xmmword ptr [ecx+edx]
00AC1225   movdqu    xmm0,xmmword ptr [edx]
00AC1229   lea       edx,[edx+10h]
00AC122C   paddd     xmm1,xmm0
```



이제 SIMD 연산은 일반화되고 있다.

Part 3. GPU 특징

CPU *Vs.* GPU



CPU 처리 과정

GPU 처리 과정

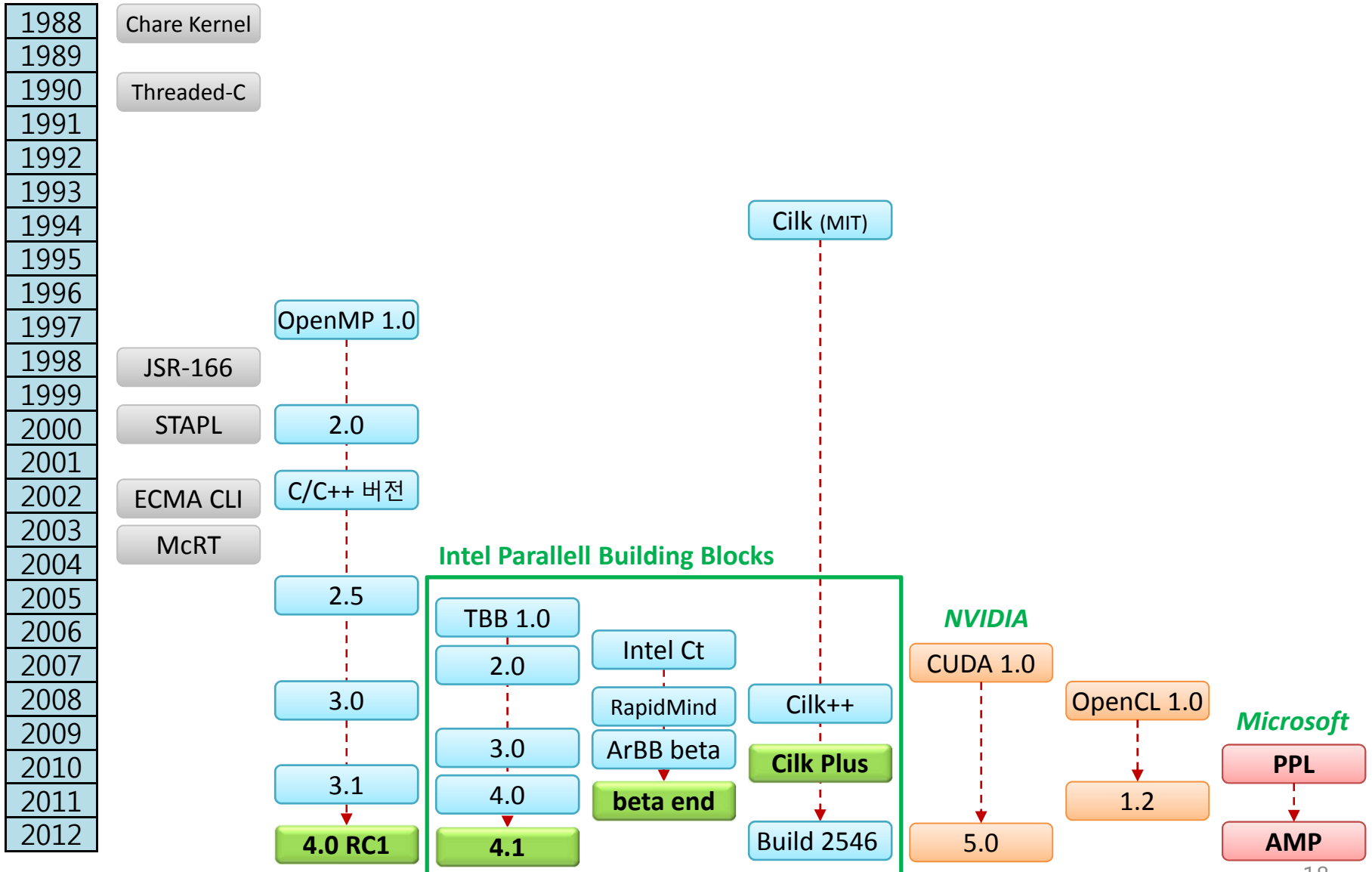
GPU 제약 사항

1. Recursion
2. 'volatile‘
3. virtual functions
4. pointers to functions
5. pointers to member functions
6. pointers in structs
7. pointers to pointers
8. goto or labeled statements
9. throw, try, catch
10. globals or statics
11. dynamic_cast or typeid
12. asm declarations
13. Varargs
14. unsupported types

    e.g. bool, char, short, long double

Part 4. 병렬화 프로그래밍 기술

병렬화 프로그래밍 기술의 발전사

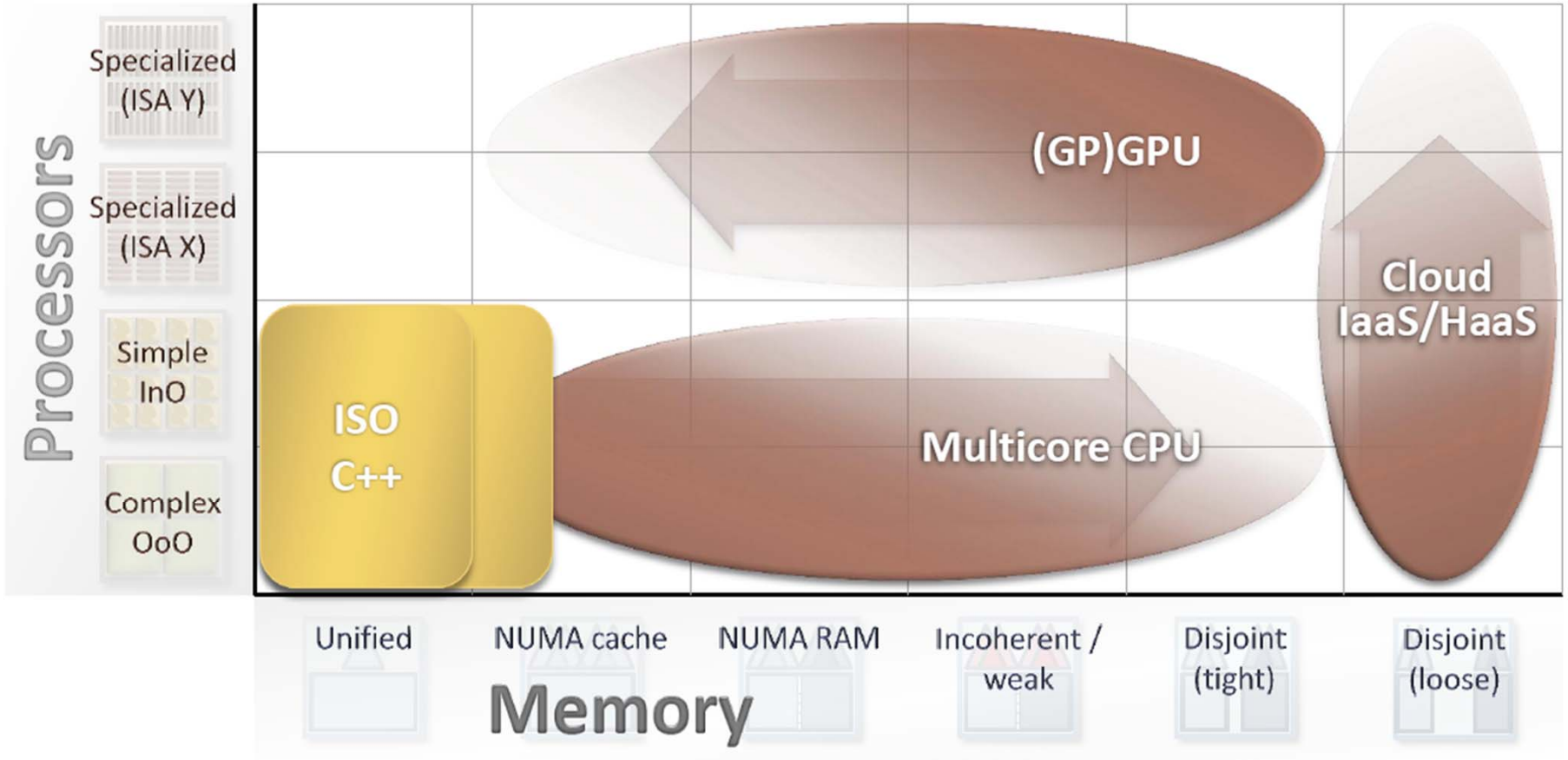


병렬화 프로그래밍 기술 *vs.* 병렬성

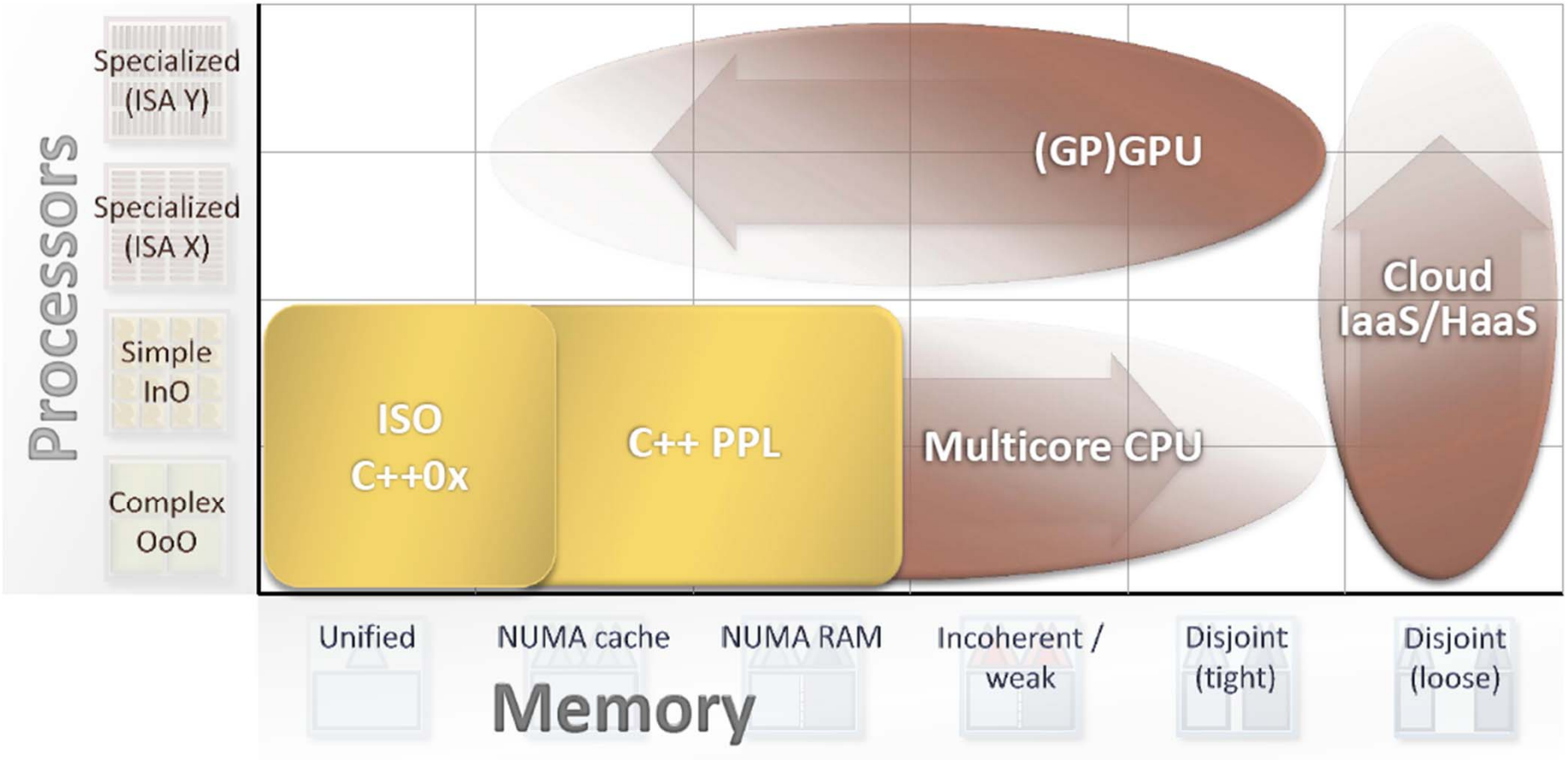
기술 /병렬성	벡터 처리	루프 병렬성	태스크 병렬성	자원
OpenMP	◌	O	O	CPU
TBB	◌	O	O	CPU
ArBB	O	O		CPU
Cilk Plus	O	O	O	CPU
PPL		O	O	CPU
CUDA	O	O		GPU
OpenCL	O	O		GPU
AMP	O	O		GPU

Part 5. AMP Portability(이식성)

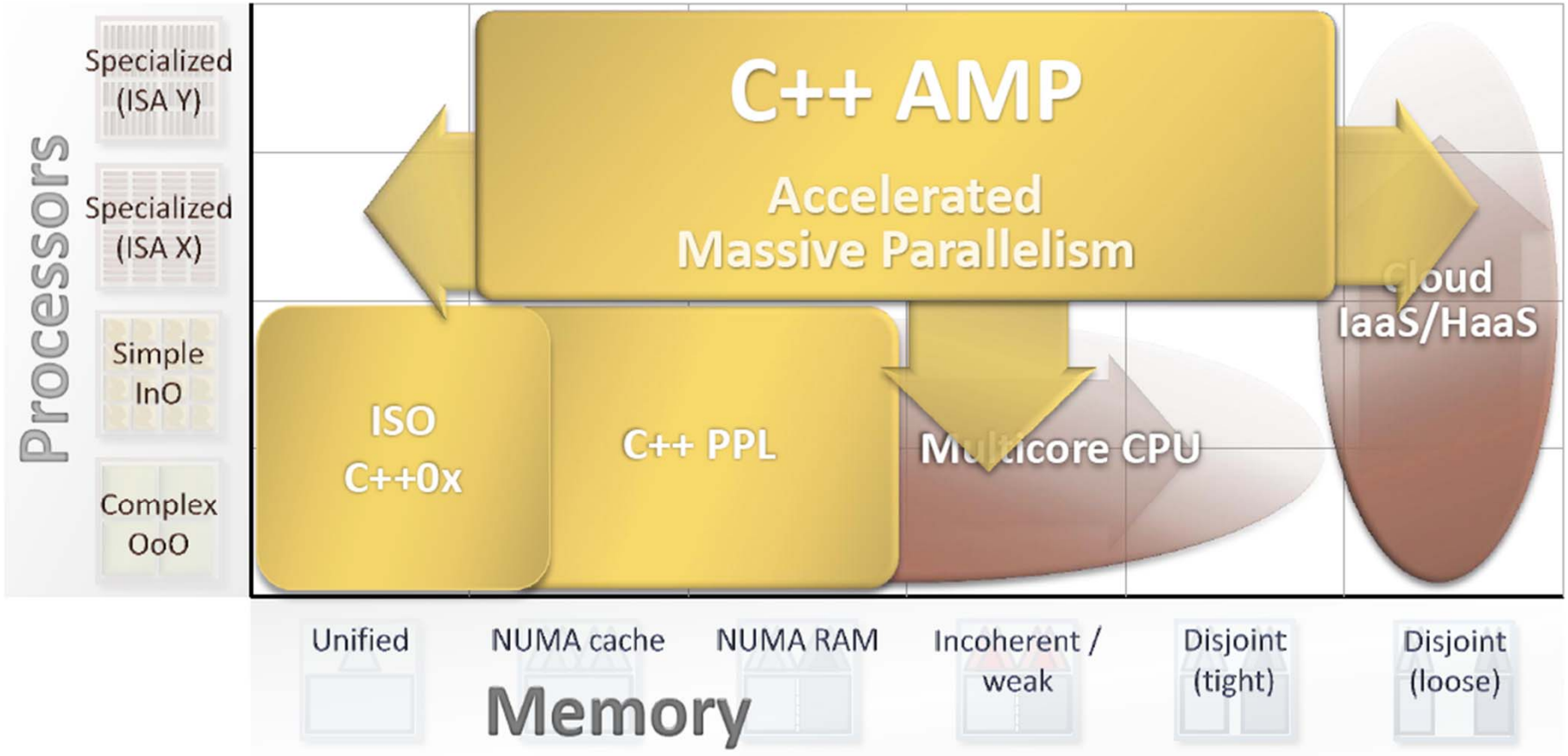
Programming Models & Languages



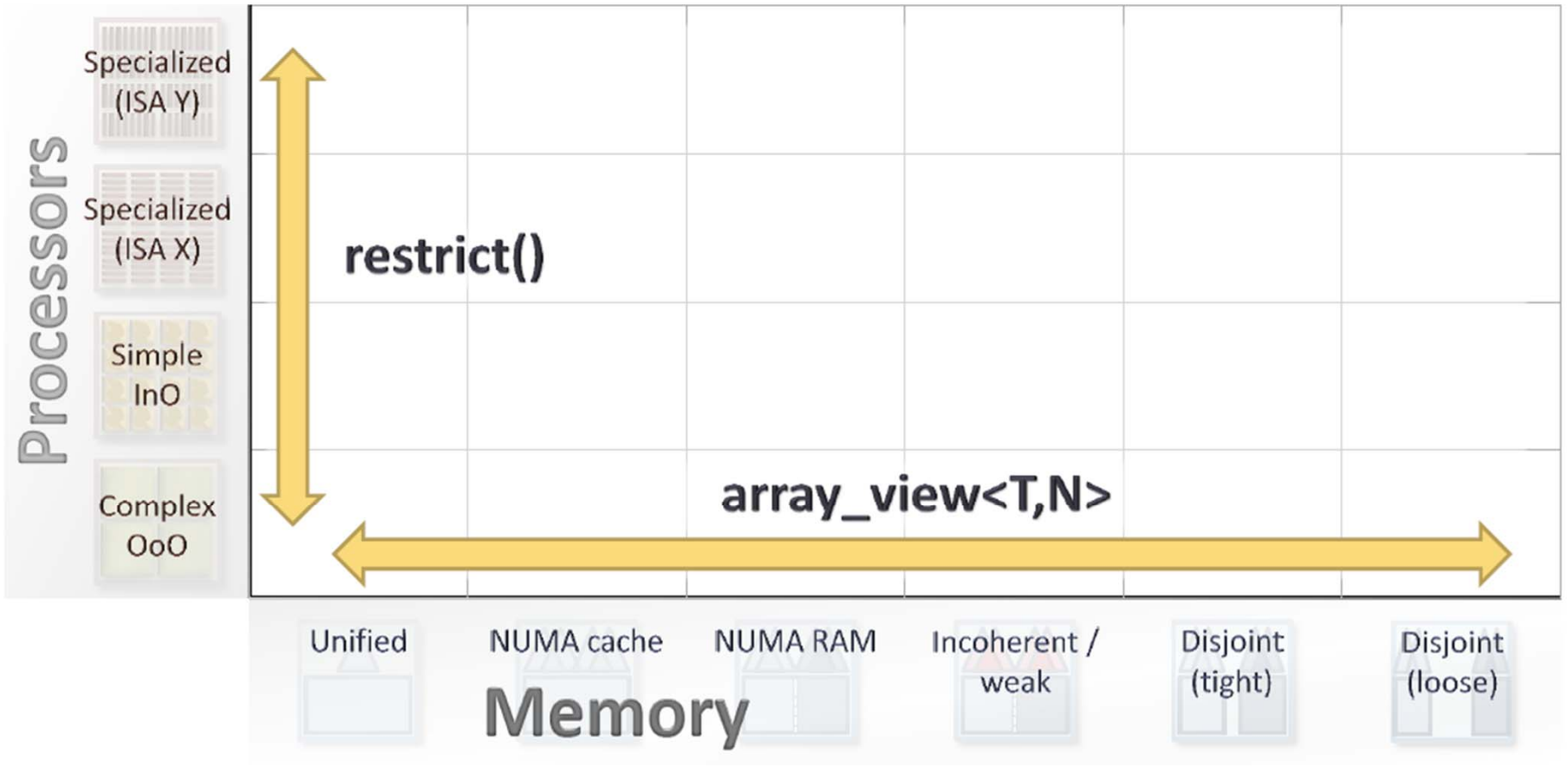
Programming Models & Languages



Programming Models & Languages



Programming Models & Languages



restrict 예약어

```
double sin(double);                          // CPU
double cos(double) restrict(amp);            // GPU
double tan(double) restrict(cpu, amp);       // CPU, GPU


parallel_for_each(av.extent, [=](index<1> idx) restrict(amp)
{
        // sin(data.analge);        // non amp restricted function

        cos(data.angle);
        tan(data.angle);
});
```

Part 6. AMP Productivity(생산성)

Productivity

Part of C++

STL-like library for

multidimensional data

Minimal API

Visual Studio integration

…

C++ to AMP

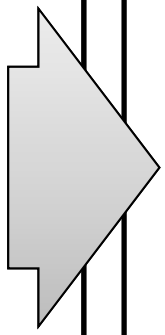
```
void AddArrays(int n, int * pA, int * pB, int * pC)
{



   for (int i=0; i<n; i++)


   {
      pC[i] = pA[i] + pB[i];
   }

}
```



```
#include <amp.h>
using namespace concurrency;

void AddArrays(int n, int * pA, int * pB, int * pC)
{
   array_view<int,1> a(n, pA);
   array_view<int,1> b(n, pB);
   array_view<int,1> sum(n, pC);

   parallel_for_each(
      sum.grid,
      [=](index<1> idx) restrict(amp)
      {
         sum[idx] = a[idx] + b[idx];
      }
   );
}
```

메모리

병렬처리

프로세서

AMP *Vs.* CUDA

```
#include <amp.h>
using namespace concurrency;

void AddArrays(int n, int * pA, int * pB, int * pC)
{
   array_view<int,1> a(n, pA);
   array_view<int,1> b(n, pB);
   array_view<int,1> sum(n, pC);

   parallel_for_each(
      sum.grid,
      [=](index<1> idx) restrict(amp)
      {
         sum[idx] = a[idx] + b[idx];
      }
   );
}
```

```
#include <cutil_inline.h>

cudaMemcpy(dev_A, InputA, size*sizeof(int),
   cudaMemcpyHostToDevice);
cudaMemcpy(dev_B, InputB, size*sizeof(int),
   cudaMemcpyHostToDevice);

VectorAdd<<<65535,512>>>(dev_A, dev_B, dev_R);

cudaMemcpy(Result, dev_R, size*sizeof(int),
   cudaMemcpyDeviceToHost);

__global__ void VectorAdd( int*a, int*b, int*c)
{
   int tid = blockIdx.x * blockDim.x+ threadIdx.x;
   c[tid] = a[tid] + b[tid];
}
```

메모리

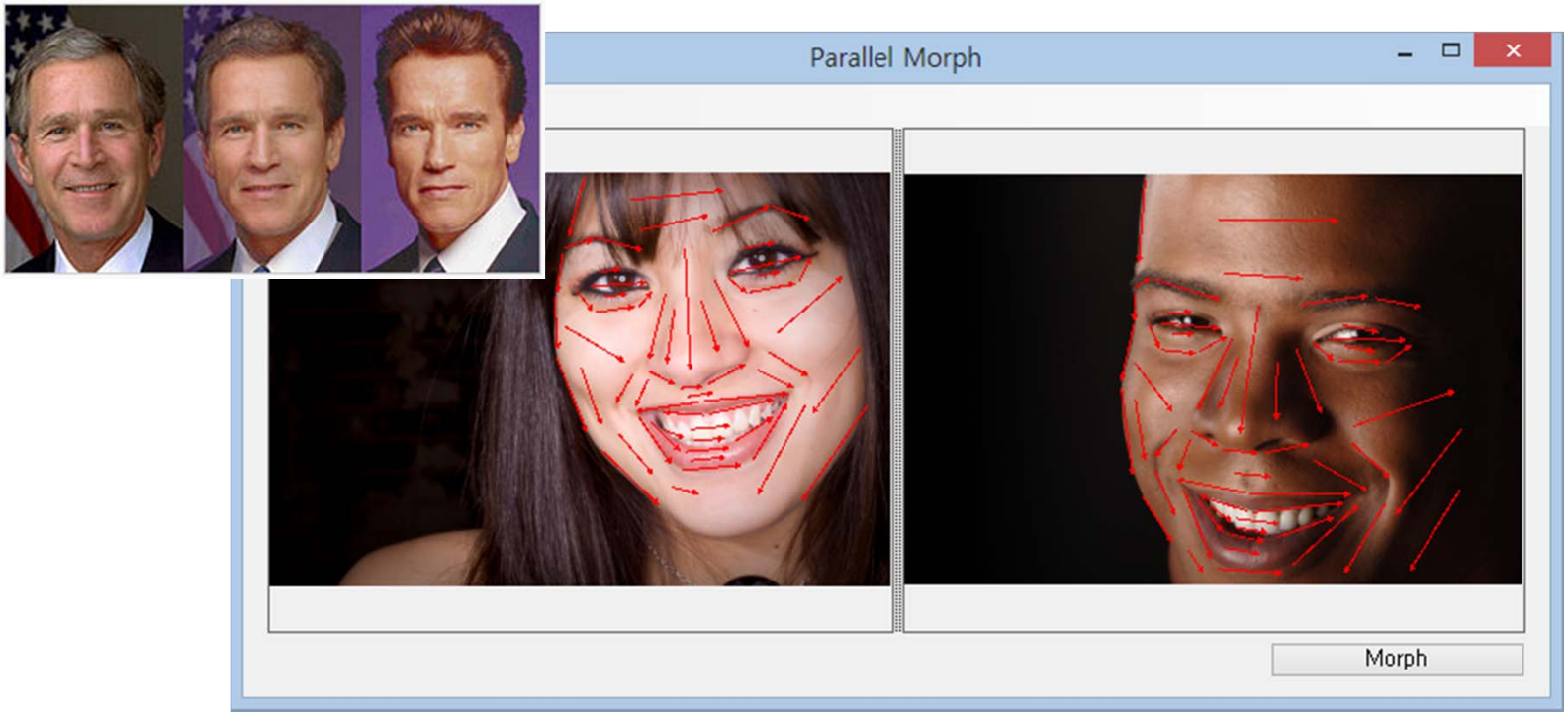
병렬 처리

연산

Part 7. AMP Performance(성능)

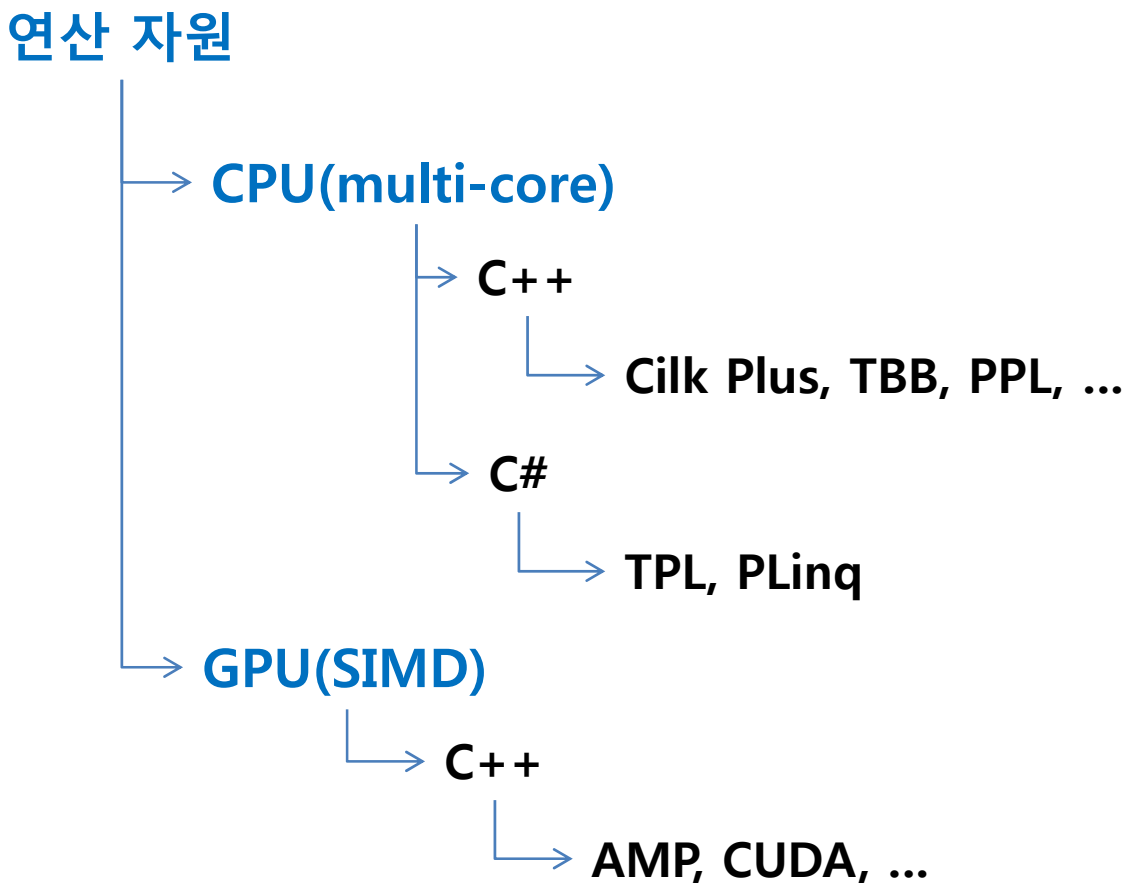
Morph

서로 다른 이미지를 자연스럽게 변화하는 과정이나 기법



Part 8. 정리

정리



이제 **CPU** 및 **GPU** 자원 활용은 **필수**다.

AMP *Vs.* CUDA

1. 이식성(Portability)

AMP > CUDA

2. 생산성(Productivity)

AMP > CUDA

3. 성능(Performance)

AMP < CUDA

AMP는

성능이 요구되는
범용 클라이언트 프로그램에 적합

또는

생산성과 **유지보수**가 요구되는
서버/클라이언트 프로그램에 적합

CUDA는

성능이 요구되는
특화된 서버/클라이언트 프로그램에 적합

참고 자료

1. 프로그래머가 몰랐던 멀티코어 CPU 이야기, 김민장

2. Heterogeneous Computing and C++ AMP

http://channel9.msdn.com/Events/AMD-Fusion-Developer-Summit/AMD-Fusion-Developer-Summit-11/KEYNOTE

3. Welcome to the Jungle

http://herbsutter.com/welcome-to-the-jungle/