CASTING SIMULATION WITH CUDA

AnyCasting Co.,Ltd

최종현 / 공학박사
최정오 / 선임연구원

Table of Contents

1. 주조해석
   (1) 주조해석이란?
   (2) AnyCasting
   (3) Equation of Solver
   (4) Poisson Equation
   (5) Poisson Equation의 종류

2. Project Procecss
   (1) Project Process Case
   (2) Project Process

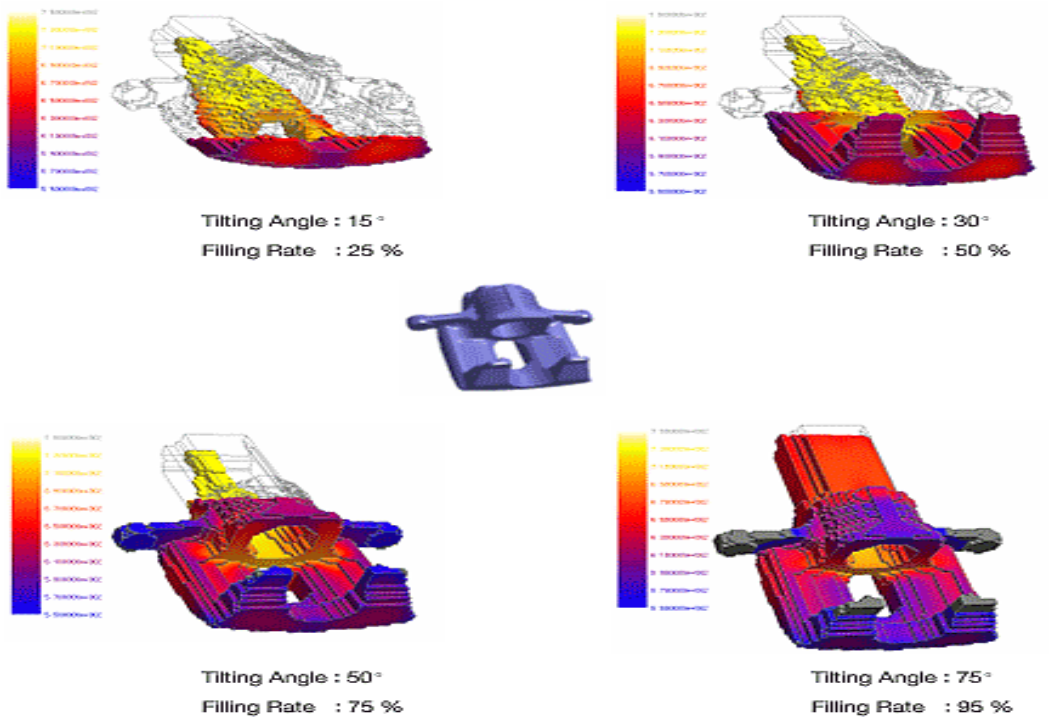
3. Pre-Test
   (1) Time – Linear Curve Fitting
   (2) Time - Exponential Curve Fitting
   (3) Time Increase - Linear Curve Fitting
   (4) Time Increase - Exponential Curve Fitting

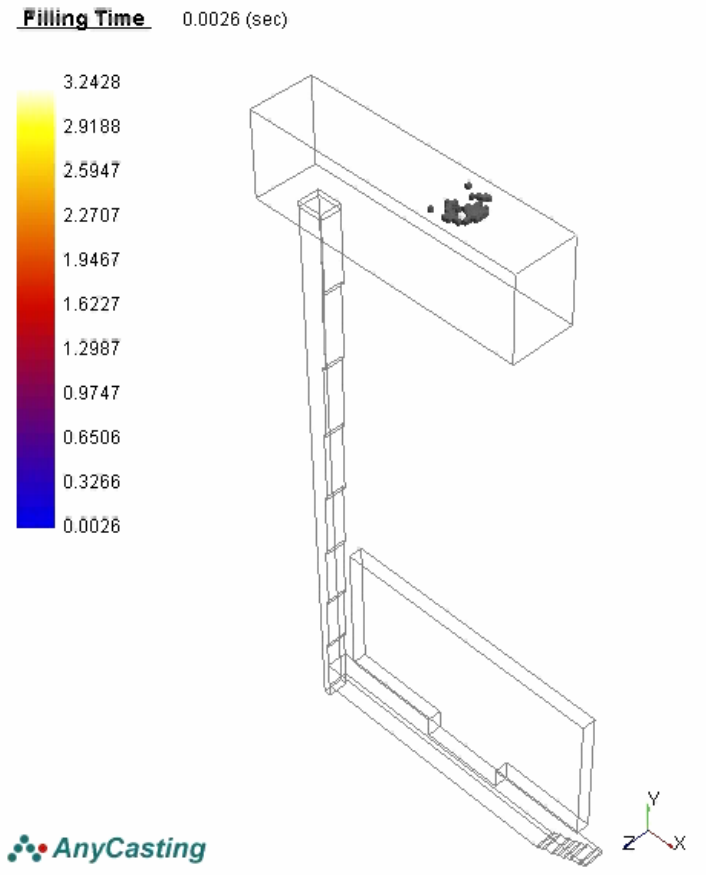
4. Work
   (1) Default Process
   (2) Special Process
   (3) Optimization

5. Why not faster & future work
   (1) Implicit Loop
   (2) Serialized Memory Access
   (3) 대용량 Memory

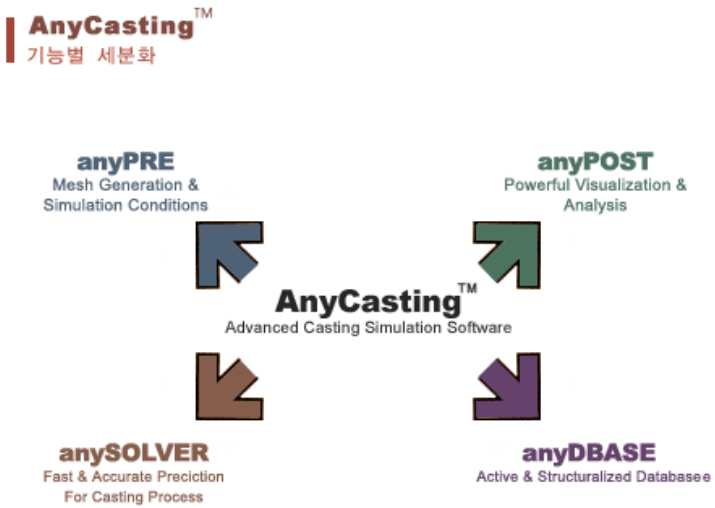
주조해석

- 주조공정시 유체 해석과 열전달 / 응고해석 연동
- 공정 중에 발생할 수 있는 결함의 형태와 위치 예측





- 주조공정 중 용탕 충전과 응고 현상을 예측
- 다양한 적용 분야
  - High Pressure Diecasting / Low Pressure Diecasting / Cyclic Casting Sand Casting / Metal Mould Casting / Investment Casting / Tilting Casting / Fluidity 히
- OpenGL 을 이용한 완전 3차원 그래픽 지원
- 타 해석 소프트웨어 대비 10-15배 이상 빠른 해석
- anyDBase / anyPre / anySolver / anyPost



- **Target of equation**
  - Variables : 4 (u, v, w, pressure)
  - Equations : 3
  - *Cannot solved*

- **Virtual Pressure**
  - u, v, w 계산 (Law of conservation of mass)
  - Pressure value ? : Poisson Equation
  - *70% of total work time*

- **Poisson Equation**
  - 공학에서 많이 사용되는 차분 방정식의 일종
  - f(x, y, z)가 0이 될 경우, 자유공간의 어떤 점에서의 특성을 표현 : Laplace's equation
  - Pressure를 찾아내는 최적화 기법으로 사용

$$\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2}\right)\varphi(x,y,z)=f(x,y,z).$$

- **Explicit Method**
  - 단순로직, 느린 계산속도
  - 각각의 값을 구하기 위해 처음부터 재계산

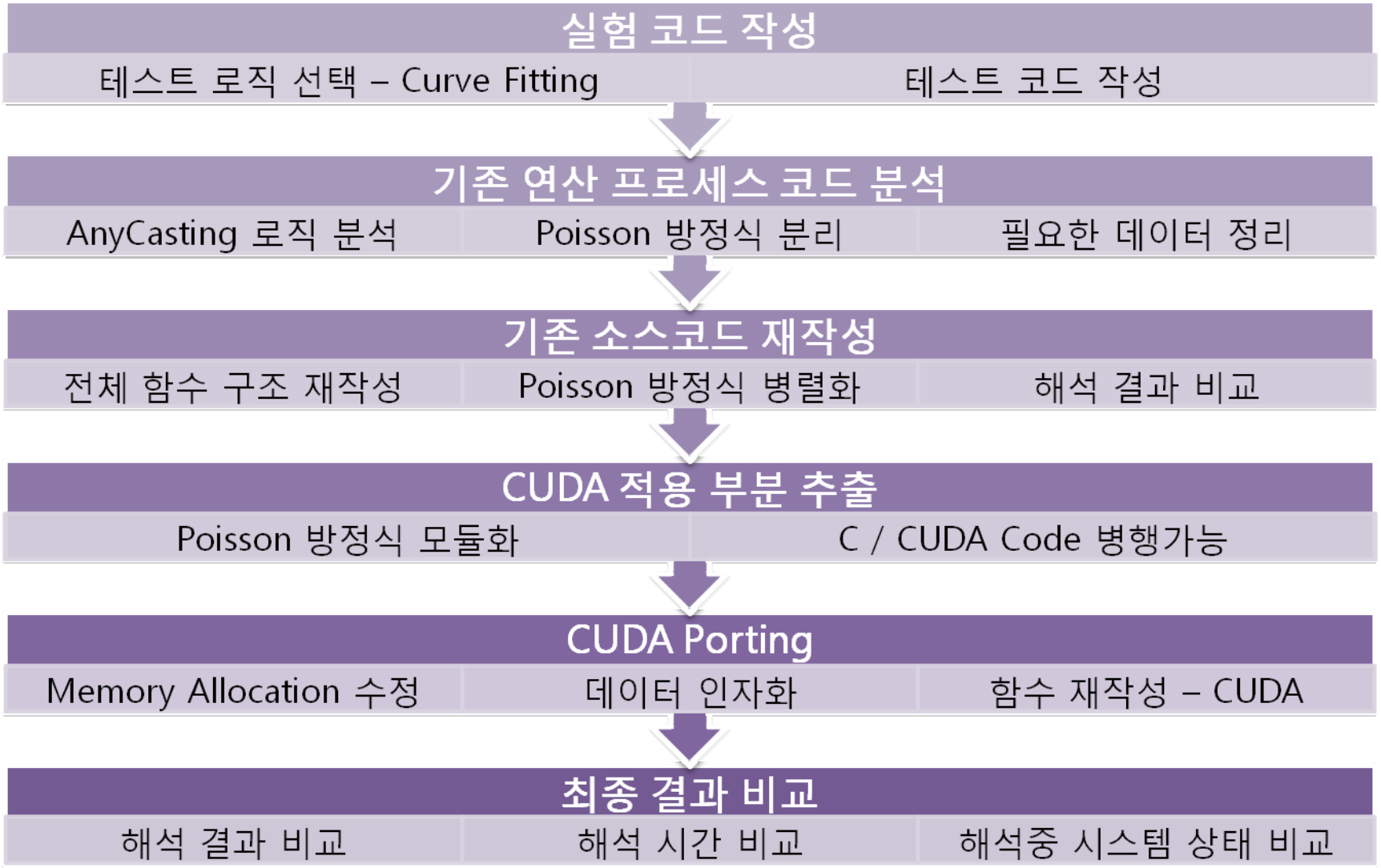
- **Implicit Method - AnyCasting**
  - 복잡한 로직, 빠른 계산속도
  - 6방향의 주변 요소 상태를 이용하여 계산
  - 행렬 연산이 많음 : 효율적인 연산 방법이 필요
  - 최소 100만번 이상, 경우에 따라 1억번 이상의 루프
    - CUDA 적용

PROJECT PROCESS

- **공학 연산 코드 분석의 어려움**

  - 전공자들의 10~20년간 연구되어온 결과물들이 반영되어 있는 지식들의 집합체
    - 해당 학문 전공자들의 깊이있는 지식
  - 수십명의 손을 거쳐온 코드 구조에 의한 난해함
    - 메모리 사용 방식, 변수형, 함수 구조
    - 개발 전문이 아닌 공학전공자들의 Programming skill 부족으로 인해, 난해한 코드 작성
  - 오래된 Code Style 및 변수 체계 해석
    - 전역 변수 선호
    - Memory 관리체계 부재

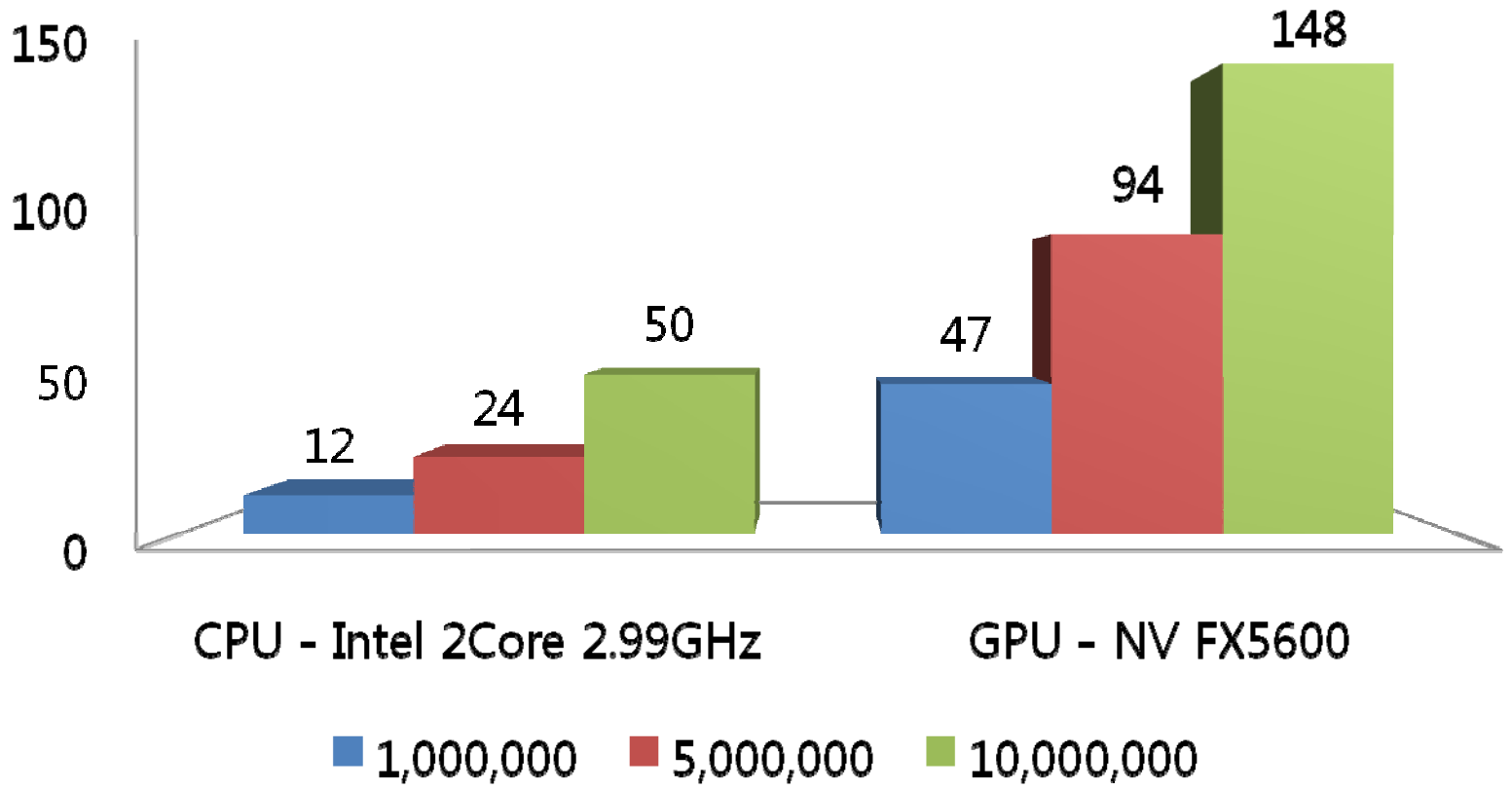
Project Process



PRE-TEST

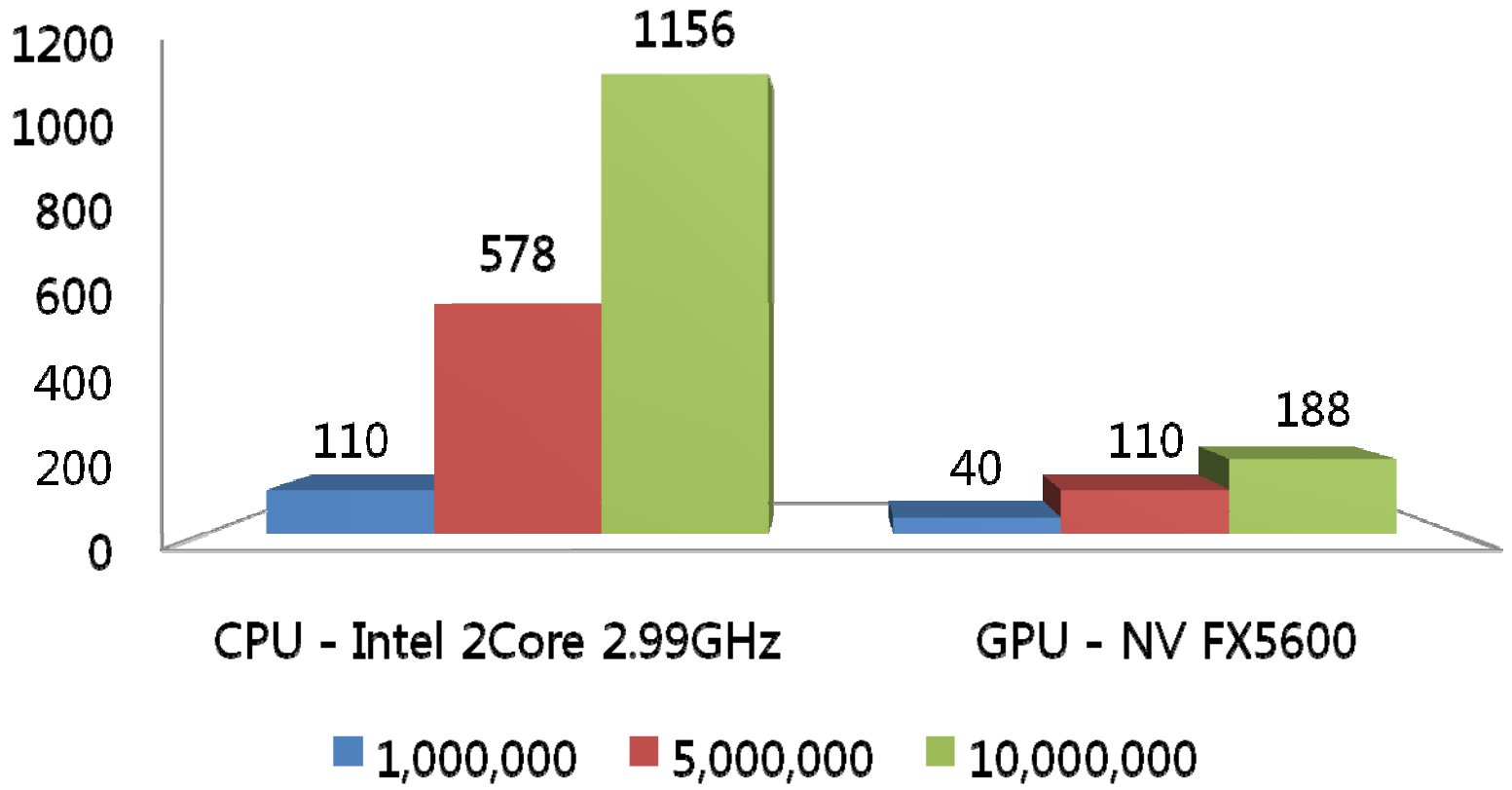
• Time – Linear Curve Fitting

단위 : 초



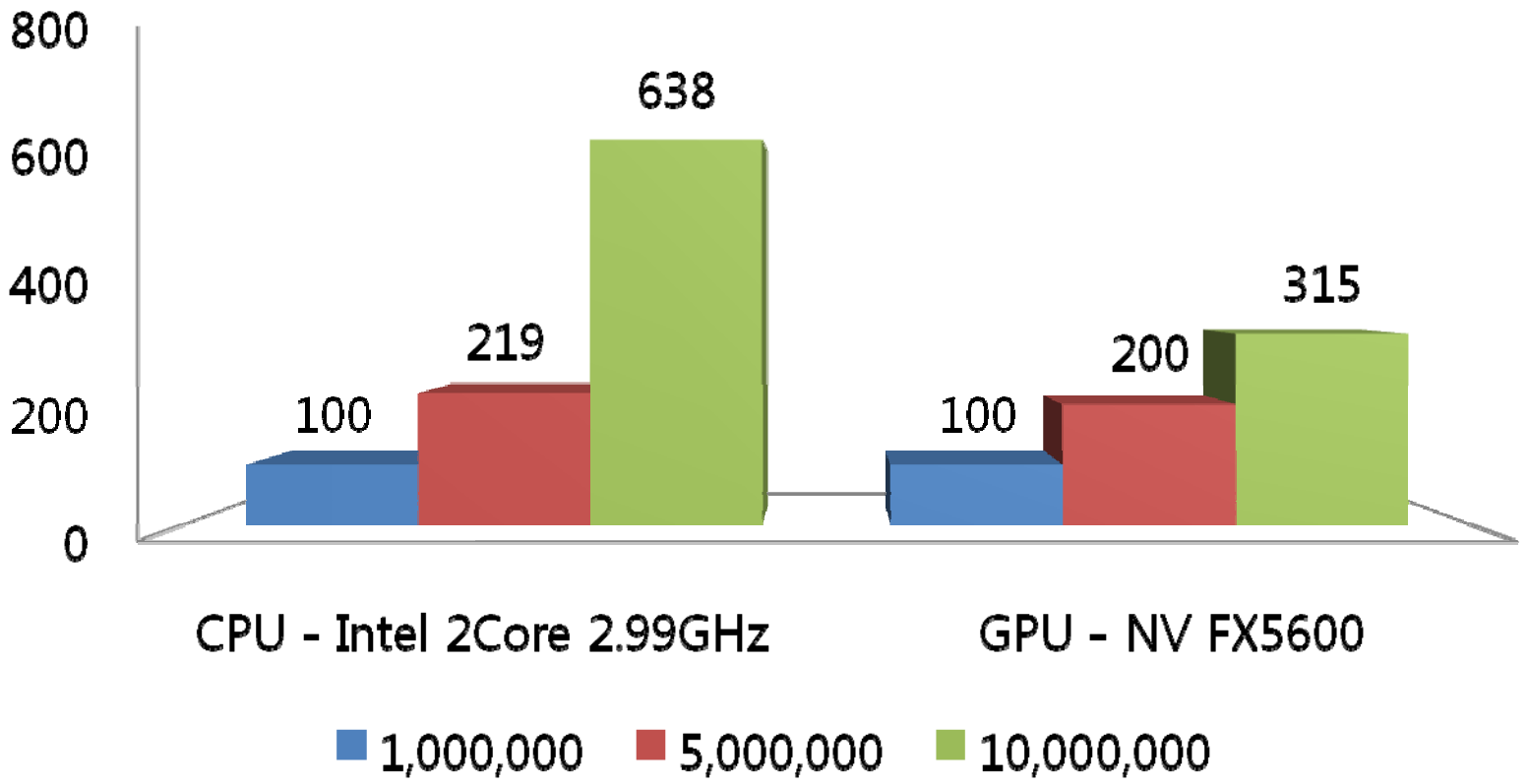
• Time – Exponential Curve Fitting

단위 : 초



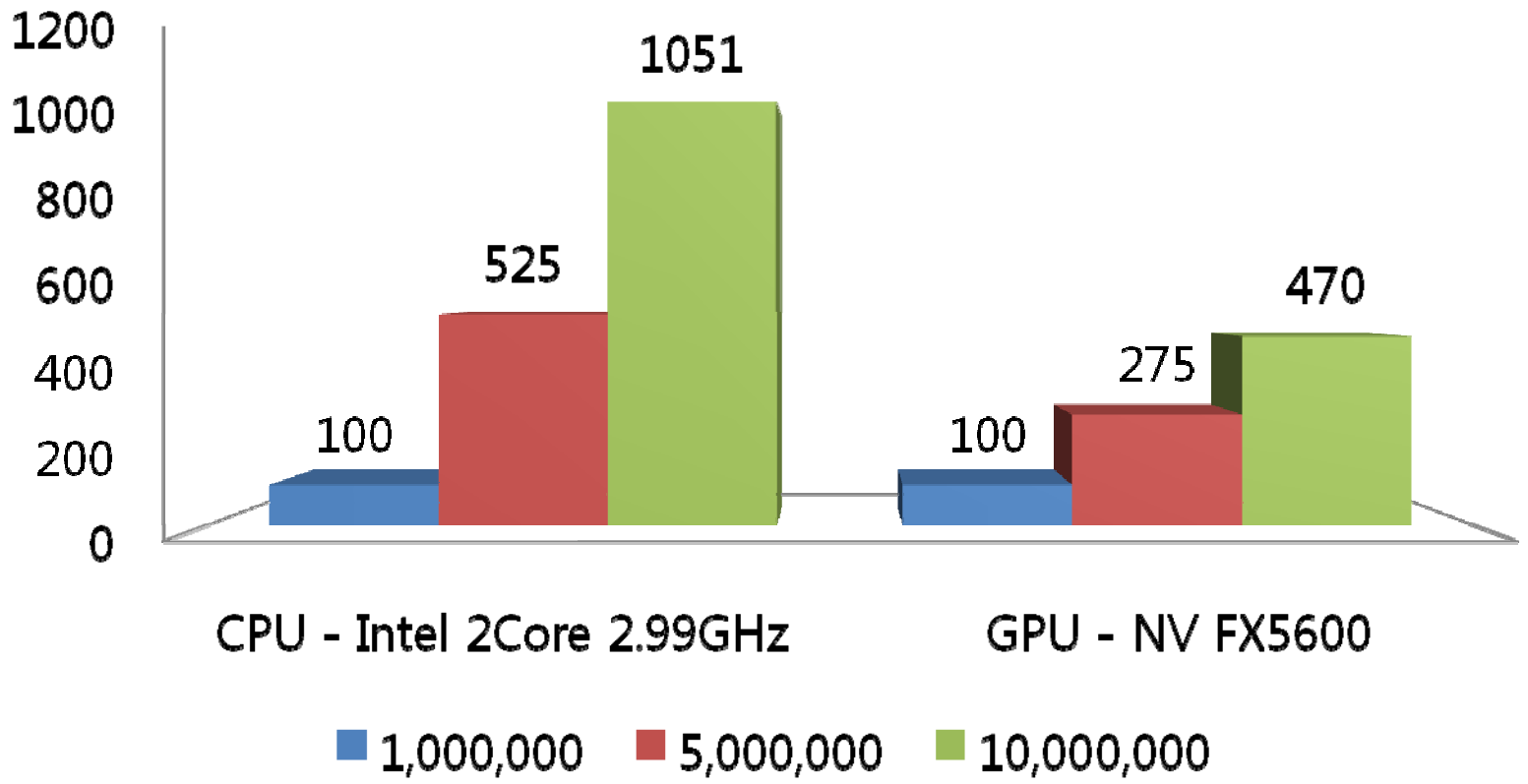
• Time Increase – Linear Curve Fitting

단위 : Percent



• Time Increase – Exponential Curve Fitting

단위 : Percent



WORK

Original

```
float *x, *y, *z …. ;

x = malloc(sizeof(float) * l);
y = malloc(sizeof(float) * m);
z = malloc(sizeof(float) * n);

void Poisson_SOR(void)
{
  int cell;

  for(int i=0; i<cell; i++)
  {
      x[i] = …..;
      y[i] = …..;
      z[i] = …..;
  }
}
```

CUDA

```
float *g_x, *g_y, *g_z …. ;

g_x = new_malloc(sizeof(float) * l);
g_y = new_malloc(sizeof(float) * m);
g_z = new_malloc(sizeof(float) * n);

// Poisson.cu 파일에 정의됨
extern void Poisson_SOR_CUDA(float *, float *, float *);

void Poisson_SOR(void)
{
  int cell, length[3]

  // X, Y, Z boundary size from cell value – multiple calculation
  length[0] = …; length[1] = …; length[2] = …;

  Poisson_SOR_CUDA(cell, length[3], g_x, sizeof_x, g_y, sizeof_y, g_z, sizeof_z);
}
```

To be continued…

- **Load A to the device**

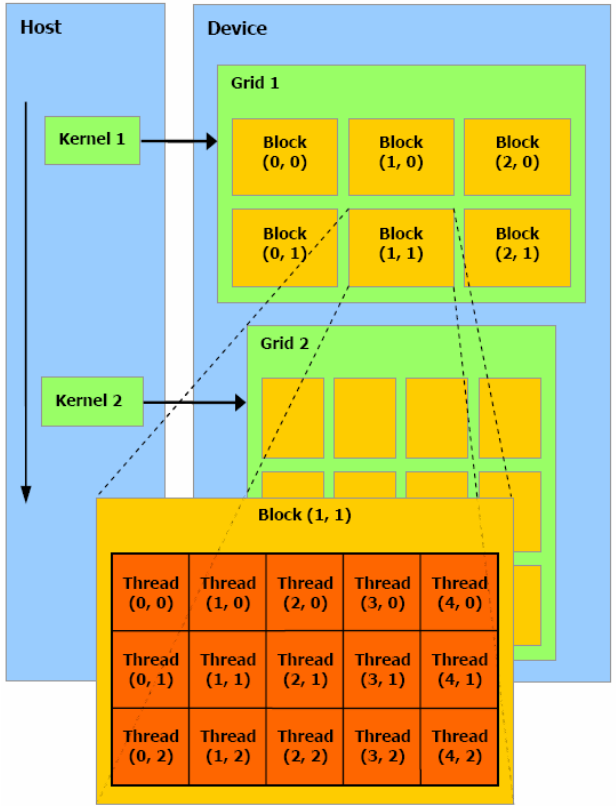
```
float *x_c = cudaMalloc(x_size);
cudaMemcpy(x_c, x, size);

float *y_c = cudaMalloc(y_size);
cudaMemcpy(y_c, y, size);

float *z_c = cudaMalloc(z_size);
cudaMemcpy(z_c, z, size);

// 결과값을 GPU로부터 받아오기 위해 준비해두는 인자
float *result_c = cudaMalloc(size);
```

- **Compute the execution configuration**

```
dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);
dim3 dimGrid(lengh[0] / dimBlock.x, length[1]/ dimBlock.y);
```

- **Launch the device computation**

```
Poissong_SOR_Device<<< dimGrid, dimBlock >>>
  (x_c, y_c, z_c, result_c);
```



- **Read result from the device**

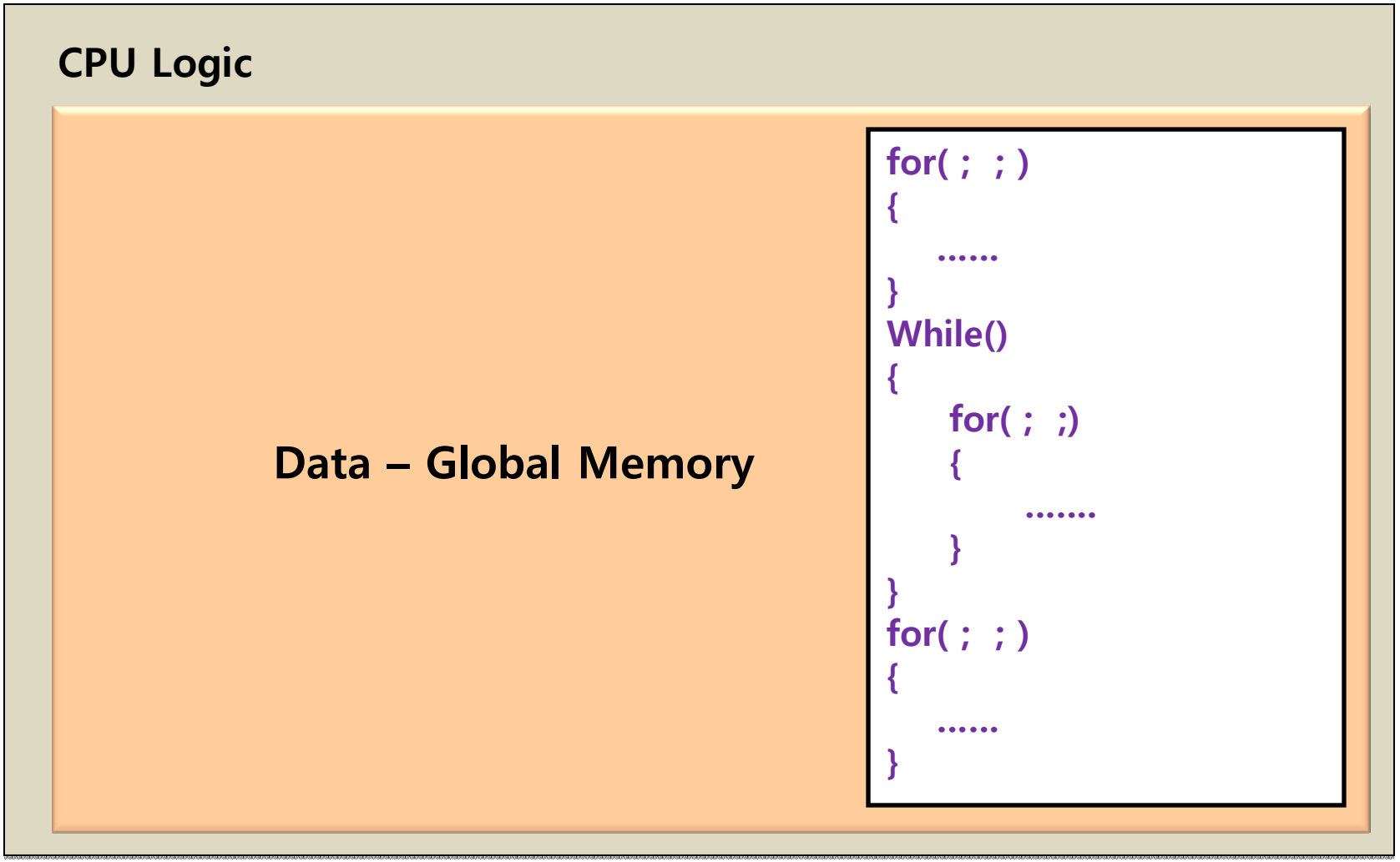
```
cudaMemcpy(result, result_c, size, cudaMemcpyDeviceToHost);
```

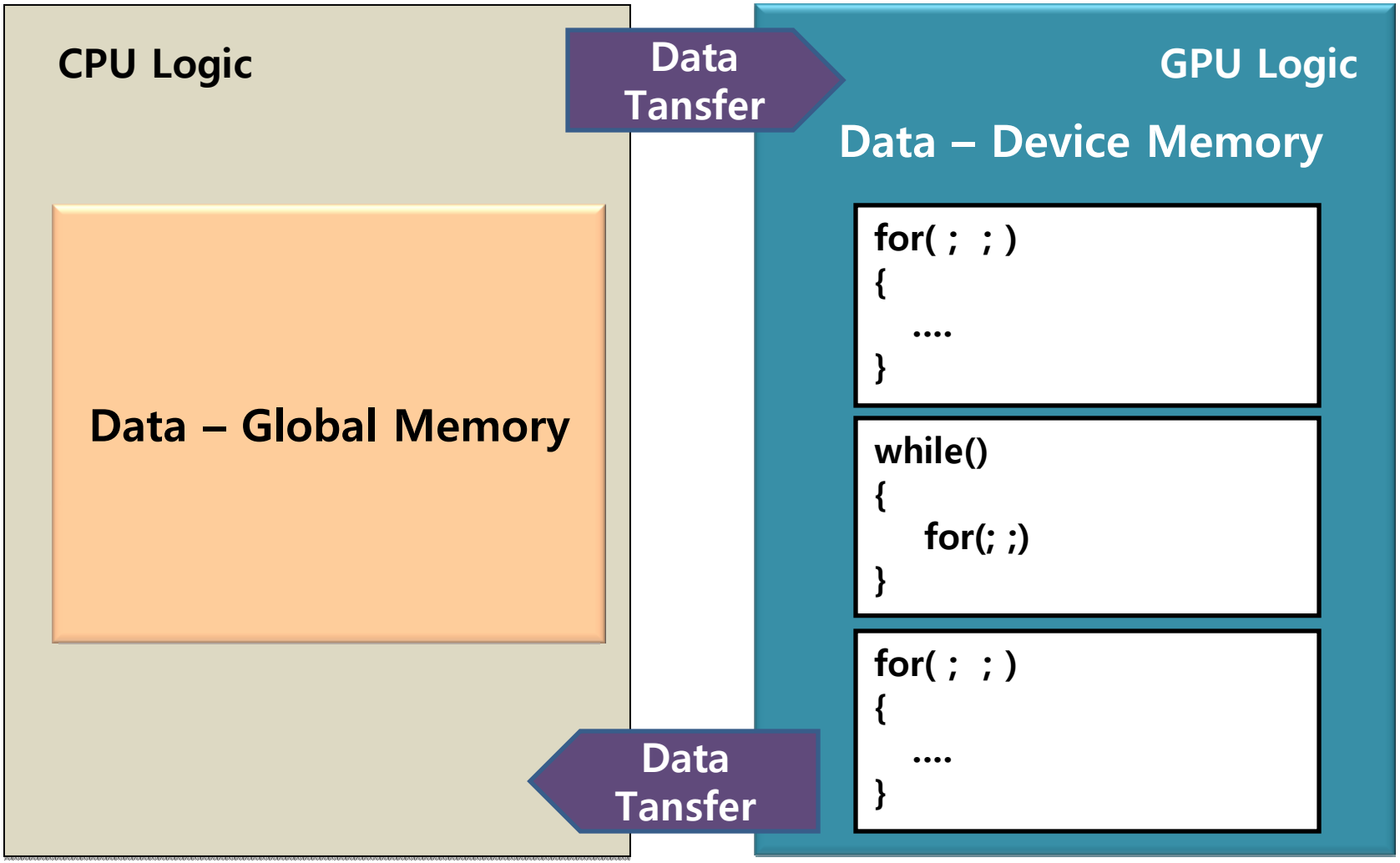
- **Free device memory**

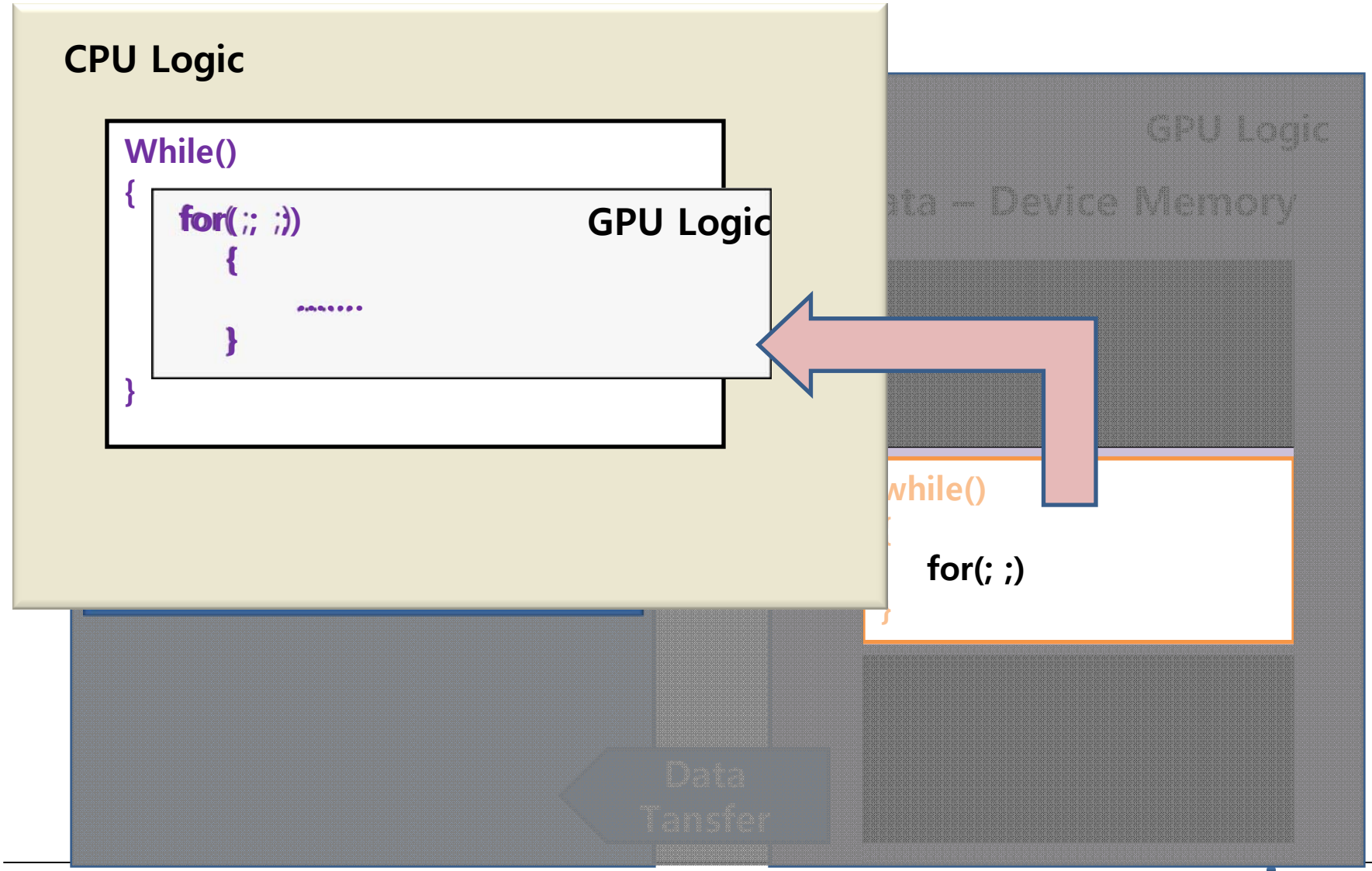
```
cudaFree(x_c);
cudaFree(y_c);
cudaFree(z_c);
cudaFree(result_c);
```

- **Device Function**

```
__global__ void Poisson_SOR_Device(int cell, float* x_c, float* y_c, float* z_c, float *result_c)
{
    int bx = blockIdx.x;          int by = blockIdx.y;
    int tx = threadIdx.x;         int ty = threadIdx.y;

    int aBegin = wA * BLOCK_SIZE * by;
    int aEnd = aBegin + wA - 1;
    int aStep = BLOCK_SIZE;
    for (int a = aBegin; a <= aEnd; a += aStep)
    {
        ...
        __syncthreads();
     }

     *result = ....;
    }
```


CPU Logic
Data – Global Memory
for(; ;)
{
.....
}
While()
{
for(; ;)
{
.....
}
}
for(; ;)
{
.....
}


CPU Logic
Data – Global Memory
Data Tansfer
GPU Logic
Data – Device Memory
for(; ;)
{
....
}
while()
{
for(; ;)
}
for(; ;)
{
....
}
Data Tansfer


CPU Logic
While()
{
for(;; ;))
{
.....
}
}
GPU Logic
GPU Logic
ata – Device Memory
while()
for(; ;)
Data
Tansfer

- **Why "while loop" in CPU?**

  - loop condition을 예측할 수 없다.
    - Thread 개수를 미리 예상할 수 없음

  - GPU Process is in blocking status about 10 – 15 seconds
    - Timeout exception 발생

  - Not Parallel Process - GPU에서 처리할 필요가 없음

- **Asynchronous Method 사용**

  - Global – Device 간의 잦은 Memory Transfer로 인한 손실 최소화
  - 3개로 분리된 Routine간에 빠른 Memory Sharing를 위한 방법
  - 수행 성능 향상 – **약 30 % 이상**

```
for (int i = 0; i < 2; ++i)
{
    cudaMemcpyAsync(inputDevPtr + i * size, hostPtr + i * size, size,
    cudaMemcpyHostToDevice, stream[i]);

    for (int i = 0; i < 2; ++i)
    {
        myKernel<<<100, 512, 0, stream[i]>>> (outputDevPtr + i * size,
        inputDevPtr + i * size, size);
    }

    for (int i = 0; i < 2; ++i)
    {
        cudaMemcpyAsync(hostPtr + i * size, outputDevPtr + i * size,
        size,cudaMemcpyDeviceToHost, stream[i]);
    }

    cudaThreadSynchronize();
}
```

WHY NOT FASTER & FUTURE WORK

- **Implicit Loop의 문제 – Parellelize 시 값이 다르게 나옴**

  - Boundary Condition check : Loop Parallelize
  - 행렬 연산에 집중하여 적용
  - 열전달 / 응고해석 등 Explicit Loop에 적용

- **Non-Serialized Memory Access**

  - Boundary Condition Check를 통하여 Memory Serialize를 강화
  - Texture Fetch 등 CUDA에서 사용하는 다른 Memory Access Method들을 적용하여 변수 접근 속도향상 필요
    - 현재 실험 결과 약 30% 향상 예상

- **대용량 Memory 전송 최적화에 실패**
  - 연산에 필요한 Data만을 분리하여 전송
  - Host Memory와 Device Memory 간의 Memory Transfer를 최소화 하기 위한 시도 필요
  - 가능한 많은 계산을 GPU에서 할수 있도록 시도 : 중간에 CPU의 로직에 Data에 간섭하는 비율을 최소화

Thank You