**수치 컴퓨팅 및 GPU 프로그래밍**

**hw4 report**

20191571 김세영

**1. 환경 명세**

Operation System: Microsoft Windows 11 Education 64-bit

Compiler: Microsoft Visual Studio Community 2019

CPU: Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz 2.90 GHz

GPU: NVIDIA GeForce GTX 1660 SUPER

OpenCL Version: 3.0

**2. CPU\_IMPLICIT\_JACOBI/CPU\_IMPLICIT\_GAUSS\_SEIDEL**

gauss-seidel method와 jacobi method의 구현 방식은 거의 유사하다. 하지만 gauss-seidel method는 float sum 을 구할 때 x[K]와 x[K\_MINUS\_1]의 값을 모두 사용하고, jacobi method는 x[K\_MINUS\_1]의 값만 사용한다.



텍스트이(가) 표시된 사진

자동 생성된 설명

****

텍스트이(가) 표시된 사진

자동 생성된 설명

gpu에서 동작할 커널 코드를 작성할 때는 float4 vector type을 사용하였지만, cpu에서는 float type을 사용한다. 따라서 float4와 같은 방식으로 float 배열의 해당 인덱스에 접근하기 위해 주어진 매크로를 사용하였다.

****

|  |  |
| --- | --- |
| CPU IMPLICIT JACOBI | CPU IMPLICIT GAUSS SEIDEL |
| **텍스트, 봉투, 문구, 명함이(가) 표시된 사진  자동 생성된 설명** | **텍스트, 명함, 봉투, 문구이(가) 표시된 사진  자동 생성된 설명** |

위의 실험 결과는 SIMULATION PARAMETERS = 0(FAST FORWARD), NUMBER OF SIMULATION STEPS = 5000에서 시뮬레이션이 끝난 결과(5000 step)를 캡처한 것이다.

두 방법 모두 implicit method이기 때문에 GPU\_IMPLICIT에 비해 시뮬레이션 결과가 안정적이다.

SIMULATION PARAMETERS=1로 변경하면 두 방법 모두 파형이 가라앉지 않은 상태에서 시뮬레이션이 끝난다.

|  |  |
| --- | --- |
| CPU IMPLICIT JACOBI | CPU IMPLICIT GAUSS SEIDEL |
|  |  |

**1) 수렴 속도**

GAUSS SEIDEL 방법은 계산 과정에서 를 사용하기 때문에 최근에 계산된 정보를 반영할 수 있어서 만 사용하는 Jacobi method보다 수렴 속도가 빠를 것으로 기대하였다.

계산에 사용되는 grid\_next(cur,prev,tmp)는 모두 float 형 배열이기 때문에 앞의 7자리까지 일치하는 경우를 수렴으로 판단하여, 초반 10 step의 수렴까지 도달하는 iter를 표로 정리하였다.

(\*SIMULATION\_PARAMETERS: 0)

|  |  |
| --- | --- |
| jacobi | gauss-seidel |
| *6* | 5 |
| 6 | 4 |
| 7 | 4 |
| 6 | 4 |
| 7 | 4 |
| 9 | 5 |
| 7 | 4 |
| 7 | 4 |
| 6 | 5 |
| 7 | 5 |

위의 결과에서 gauss-seidel method가 jacobi method보다 수렴속도가 빠른 것을 확인할 수 있다.

**2) 계산 값**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| jacobi | | | gauss-seidel | |
| grid\_next | |현재 grid\_next- 이전 grid\_next| | | grid\_next | |현재 grid\_next- 이전 grid\_next| |
| 10.7685 | |  | 10.7691 |  |
| 10.7654 | | 0.0031 | 10.7659 | 0.0032 |
| 10.7623 | | 0.0031 | 10.7628 | 0.0031 |
| 10.7591 | | 0.0032 | 10.7597 | 0.0031 |
| 10.756 | | 0.0031 | 10.7566 | 0.0031 |
| 10.7529 | | 0.0031 | 10.7534 | 0.0032 |
| 10.7498 | | 0.0031 | 10.7503 | 0.0031 |
| 10.7467 | | 0.0031 | 10.7472 | 0.0031 |
| 10.7435 | | 0.0032 | 10.7441 | 0.0031 |

위의 표는 SIMULATION PARAMETERS=1, NUMBER OF SIMULATION STEPS = 5000 일 때 두 시뮬레이션의 마지막 9 step의 grid\_next[Y(64 \* n\_nodes\_1d + 64)], 현재 grid\_next의 값에서 이전 step의 grid\_next 값 을 뺀 것의 절댓값을 표로 정리한 것이다.

CPU\_JACOBI(혹은 GaussSeidel)\_DEBUG를 1로 한 경우 jacobi method는 jc.txt, gauss-seidel method는 gs.txt에 출력된다.

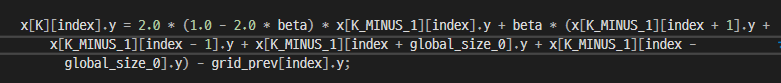
위의 결과를 보면, gauss-seidel의 grid\_next 값이 조금 더 크지만 두 방식의 값의 변화량은 비슷하다고 할 수 있다. 또한 실험결과를 눈으로 확인했을 때 두 시뮬레이션 방법의 결과 차이를 발견할 수 없었다.

**3) 수행시간**

수행시간은 Jacobi method가 1.291ms, Gauss-Seidel이 1.738ms으로 Jacobi method 가 더 빠르다. Jacobi는 sum을 계산할 때 x[K\_MINUS\_1]의 값들만 더하지만 Gauss-Seidel는 x[K\_MINUS\_1]와 x[K]에 모두 접근하기 때문에 global memory access 시간이 더 오래 걸린 것으로 추정할 수 있다.

**3. GPU\_IMPLICIT/GPU\_EXPLICIT**

explicit method는 이전 프레임에서 구한 값들을 직접 식에 대입하여 계산한다. central difference는 계산 당 의 오차가 나온다.

****

|  |  |
| --- | --- |
| GPU\_EXPLICIT | GPU\_IMPLICIT |
| **텍스트이(가) 표시된 사진  자동 생성된 설명** | **텍스트, 명함, 봉투, 문구이(가) 표시된 사진  자동 생성된 설명** |

위의 실험 결과는 SIMULATION PARAMETERS = 0, NUMBER OF SIMULATION STEPS = 5000 에서 시뮬레이션이 끝난 결과(5000 step)를 캡처한 것이다. workgroup size는 다음과 같다.



explicit method는 오차가 누적되었기 때문에 implicit method보다 결과가 불안정하다.

SIMULATION PARAMETERS=1로 변경하면 두 방법 모두 파형이 가라앉지 않은 상태에서 시뮬레이션이 끝나고, 두 방식 모두 CPU에서 구현했던 implicit method와 유사한 결과가 나온다.

|  |  |
| --- | --- |
| GPU\_EXPLICIT | GPU\_IMPLICIT |
|  | **텍스트, 봉투, 명함, 벡터그래픽이(가) 표시된 사진  자동 생성된 설명** |

explicit method의 수행시간은 1.483ms, implicit method의 수행시간은 1.725ms로 explicit method가 더 빠르다. 그러나 explicit method는 계산 과정에서 오차가 누적되기 때문에 계산 결과가 정답과 다른 경우가 생긴다. 위에 설명한 SIMULATION PARAMETERS = 0, NUMBER OF SIMULATION STEPS = 5000가 이에 해당된다.

gpu implicit method의 수행시간을 줄이기 위해 여러 work group size에서 동일한 시뮬레이션을 진행했다.

(단위: ms)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| (32,8) | (32,16) | (32,32) | (64,8) | (64,16) |
| 1.725 | 1.704 | 1.707 | 1.685 | 1.663 |

(64,16)일 때의 수행시간이 제일 적다.