**분병컴 과제1 소프트웨어학과 202126891 고기호**

**1. 코드 설명**

**1.1 main**

코드 파일에는 main.c, koh.h, koh.c가 있다. main함수에서는 실행할 때 받은 명령 인자를 통해 입력을 받는다. 여기서는 전역 격자 크기인 nx, ny와 프로세스 격자 크기인 px, py 그리고 확산 계수와 스텝을 차례로 입력된다. 여기서 mpi 명령을 실행할 때의 -np의 갯수와 px\*py값이 다르면 프로그램이 정지한다. 또한, nx는 px의 배수이고, ny는 py의 배수가 아니면 정지하게 만들어, 모든 프로세스가 담당하는 크기를 같다고 가정했다. 이후 이 값들은 koh.h파일에 있는 MPI\_info 구조체를 통해 저장된다. 참고로 구조체 안에 있는 local\_xsize, local\_ysize(이후 local\_size)와 coord\_px, coord\_py(이후 coord)는 각각 한 프로세스가 전역 격자에서 담당하고 있는 사이즈와 한 프로세스의 좌표를 나타낸다. 이 구조체를 만든 이유는 여러 함수로 정보를 쉽게 전달하기 위함이다.

이후 T\_old와 T\_new 배열을 동적 할당을 통한 2차원으로 만들었다. 이때 배열의 사이즈는 위에서 계산한 local\_size에 2를 더한 크기만큼 할당 했는데, 이는 다른 프로세스로부터 받은 경계값을 저장하기 위함이다. 그래서 실제 본인 프로세스의 값은 (1,1)에서부터 저장된다. 온도의 경우 본인의 인접 프로세스에게서 받아서 계산해야 하므로, arr\_x와 arr\_y를 통해 기본적인 상하좌우 탐색을 위한 배열을 만들었다.

**1.2 check\_bound()**

check\_bound 함수는 전체 경계를 확인해, 본인 프로세스 기준으로 상하좌우 중 어느 곳이 전역 격자의 끝인지 확인하고 저장한다. 이거를 koh.c에 있는 send\_dir, recv\_dir(이후 dir)에 저장한다. 이 값이 false이면 그 부분이 막힌 부분 즉 단열이 필요한 부분이다. Send와 recv로 나눠 놓은 이유는 데드락을 피하기 위해서다. 보내는 프로세스는 상좌하우 순서로 보내고, 받는 프로세스는 하우상좌 순서로 받는다.

**1.3 insulate(), exchange\_bound()**

이후 본격적인 계산을 시작한다. 먼저 exchange\_bound 함수를 통해 경계를 교환한다. 이 함수는 단열, 정보교환을 담당하는 핵심 함수이다.

단열은 insulate함수를 통해 진행한다. 아까 만들어 놓은 dir을 통해 상하좌우 어디가 단열을 부분인지 확인을 하고, 그 부분은 그 바로 앞의 값을 복사해 저장한다.

이후 exchange\_bound에서는 통신이 시작된다. 처음에는 홀수 rank가 먼저 send하고 짝수 rank가 recv하는 것으로 코드를 짰었는데 프로세스 수가 짝수일 때, 짝수인 프로세스가 짝수인 rank로 보내는 데드락이 발생했다. 그래서 이를 해결하기 위해 저장 해놓은 coord를 통해 x,y값을 더하고, 나머지 연산(coord%2)을 통해, 나머지가 0인 프로세스가 먼저 recv를 하고, 1인 프로세스는 send를 해서 데드락을 방지했다.

**1.3.1 send(), get\_dest(), make\_message()**

나머지가 0, 1인 프로세스들의 코드는 서로 send, recv하는 순서만 다르고 동일하므로, send와 recv에 대해서 설명하겠다. 먼저 send에서는 dir을 통해 보낼 대상이 있는지, 즉 자기가 이번에 보낼 대상이 끝인지 확인을 하고, 끝이면 send를 진행하지 않는다. 이는 위에서 이미 insulate함수를 통해 단열을 했기 때문이다. get\_dest 함수는 i로 방향과 본인의 rank를 통해 목적지 rank값을 반환한다.

Send에서는 i에 따라 방향이 바뀐다. 0부터 시작해서 3까지 각각 상좌우하 방향을 담당한다. 방향을 이렇게 구성한 이유는, 상하와 좌우일 때 서로 보내는 메시지의 크기가 다르기 때문에 %2 연산을 통해 메시지 사이즈를 구분하기 위해서다. 0일 때는 상하로 보내는 local\_xsize가 활용되고, 1일 때는 좌우로 보내는 local\_ysize가 활용된다. 여기서 count는 이 메시지의 사이즈이다. 이후 make\_message를 통해 보낼 메시지를 만든다. make\_message 함수는 i를 통해 방향을, 그리고 T\_old배열을 통해 다른 프로레스에게 보낼 값을 얻어 message에 저장한다. 이후 MPI\_Send 함수를 통해 전송한다. 보낼 때의 tag는 자신이 보내는 방향을 나타낸다.

**1.3.2 recv(), get\_origin(), read\_message()**

recv는 send함수와 비슷하다. 먼저 똑같이 dir을 통해 보낼 대상이 있는지 없는지 확인을 하고 get\_origin함수를 통해 어디로부터 받을지 방향을 정한다. 여기서 방향은 좌상우하 인데, 이는 i가 0부터 시작했을 때이다. recv에서 i는 3부터 시작하므로, 실질적인 동작 방향은 하우상좌이다. 이후, %2연산을 통해 상하, 좌우일 때의 메시지 크기와 count를 설정하고, MPI\_Recv를 통해 메시지를 받는다. 여기서 태그는 3-i로 해서, 보낼 때의 i와 순서를 맞추었다. 예를 들어 send에서 i가 1이면, 왼쪽으로 보내는 것이고, recv에서는 i가 2이면 오른쪽 것을 받아 서로 메시지를 주고 받는 것이다. 이후 read\_message를 통해 메시지를 해석해 방향에 맞게 T\_old에 저장한다.

**1.4 계산**

exchange\_bound를 통해 단열과 경계값 등 계산에 필요한 값들을 모두 받았으면 이제 계산을 시작한다. 아까 탐색을 위한 방향 arr\_x, arr\_y를 통해 동서남북으로 탐색하며, 이 값을 arr 배열에 저장한다. 한 동서남북 탐색이 끝나면, 열 계산 공식을 통해 열을 계산해 T\_new에 새로운 값을 저장한다.

이후 한 스텝에 대해 T\_new의 계산이 끝나면, T\_old에 있는 이전의 경계 값(전달 받은 값, 단열)들을 초기화한다. 이후 T\_old로 스왑을 한다.

**1.5 gather\_total\_heat(), 마무리**

이후 gather\_total\_heat을 통해 전체 합을 계산한다. 각 프로세스는 T\_old에 있는 최신 값들을 local\_size에 있는 값들만(경계 값 제외) 더한다. T\_old가 최신인 이유는 계산을 하고 마지막에 T\_new와 T\_old를 스왑했기 때문이다. 이후, master rank(rank=0)이 다른 rank 들로부터 차례로 sum을 받아 합쳐 결과를 출력한다. 이때 각 프로세스의 max값도 전송을 해서 100스텝 이후의 최대 max 온도를 출력한다.

**2. Build 및 Run command**

빌드할 때는 mpicc를 써서 빌드했는데, 내가 짠 코드는 main.c, koh.h, koh.c 메인파일과소스파일로 나누었기 때문에, *mpicc -o main main.c koh.c* 를 통해 컴파일 했다. 그 후 실행할 때는 *mpirun -np 16 ./main 4096 4096 4 4 0.22 1000* 명령어를 사용했다. 차례로 nx, ny, px, py, alpha, step을 나타낸다.

**3. 참고한 사이트**

참고한 사이트는 [[1]](#footnote-1) 이다. 여기서 mpi코드를 보며 공부를 하고 코드를 실행해보고 과제를 시작했다.

**4. Generative AI**

첫 시험 코드, send와 recv없는 버전, 을 짜고 segmentation fault가 나서 지피티를 통해 답을 코드에서 잘못된 메모리 접근(배열)이 있다는것을 발견하고 수정했다.

맨 처음에 정보 교환을 rank 홀수, 짝수 이렇게 구분한 것도 지피티를 통해 보통 이렇게 한다고 답을 줘서 이렇게 진행했었는데, 막상 데드락 문제가 발생해서 나중에는 직접 현재의 체커보드 스타일로 변경했다.

마지막에 합산을 하기 전에 동기화를 위해서, 다른 랭크들이 모두 기다리게 하는 명령어가 있는지 궁금해서 지피티를 통해 MPI\_Barrier의 존재를 알았다. 하지만 이거를 빼고 해도 문제가 발생하지 않아서, 빼고 진행했다.

처음 코드를 완성했을 때는 단열의 존재를 몰랐다. 교수님이 총 합이 100이 될 것이다라고 말씀하신 것만 기억하고, 또한 단열에 관해서는 알지 못했다. 그래서 실행시켜보니 총 합이 0이 되었는데, 왜 이런지 계속 고민을 하다가 지피티에게 물어봐서 단열의 존재를 알고, 과제 설명에 있는 단열의 내용을 확인하고, 그 후에 insulate함수를 추가해서 현재 코드를 완성했다.

**5. 실험 결과 표 + 스크린샷**

실험에서 px,py는 각각 4,4로 설정하고, alpha는 0.22로 진행했다.

실행결과(스크린샷)에서 rank 0 전체 열 합계가 전체 온도 합이고, 전체 max가 전체 최고 온도이다.

**5.1 열점 고정(왼쪽 모서리 끝)**

|  |  |  |  |
| --- | --- | --- | --- |
| Nx | Ny | Step | 스크린샷 |
| 512 | 512 | 100 |  |
| 1024 | 1024 | 200 |  |
| 2048 | 2048 | 400 |  |
| 4096 | 4096 | 1000 |  |

**5.2 프로세스 수 변경**

이번엔 프로세스 수를 64개 8\*8로 바꾸고 실행시켜 보았다.

|  |  |  |  |
| --- | --- | --- | --- |
| Nx | Ny | Step | 스크린샷 |
| 512 | 512 | 100 |  |
| 1024 | 1024 | 200 |  |
| 2048 | 2048 | 400 |  |
| 4096 | 4096 | 1000 |  |

실행 결과 프로세스 수와 상관 없어 똑같은 결과가 나오는 것을 확인 할 수 있다.

**5.3 열점 변경**

여기서 r은 rank이고 tx,ty는 그 rank내에서의 좌표이다.

코드에서 보면 main.c에서 밑에 사진과 같은 부분을 변경해서 실행했다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nx | Ny | r | tx | ty | Step | 스크린샷 |
| 512 | 512 | 3 | 1 | 2 | 100 |  |
| 4096 | 4096 | 12 | 3 | 5 | 1000 |  |
| 4096 | 4096 | 9 | 10 | 11 | 1000 |  |

1. https://github.com/mpitutorial/mpitutorial/tree/gh-pages/tutorials/mpi-send-and-receive/code [↑](#footnote-ref-1)