[Lab-3] POSIX Thread

소프트웨어학과 201720707 나용성

1. Describe how to parallelize your game of life

   game of life의 구현은 m_Grid에 저장된 generation i-1에서의 셀들의 값과 game of life 룰을 이용해 m_Temp에 generation i의 셀들의 값을 구하여 저장한 뒤 모든 셀의 값을 구하면 generation i+1로 넘어가기 전에 m_Temp에 저장된 generation i의 셀들의 값을 m_Grid에 모두 저장하는 과정을 통해 구현된다.

```
if (nprocs == 0) {
  // Running with your sequential version
  while(g_GameOfLifeGrid->getGens()){
    for(int i = 0; i < rows; i++){
      for(int j = 0; j < cols; j++){
    if(g_GameOfLifeGrid->isLive(i, j)){
      int NumOfNeighbors = g_GameOfLifeGrid->getNumOfNeighbors(i,j);
      if(NumOfNeighbors == 2 || NumOfNeighbors == 3)
        g_GameOfLifeGrid->live(i,j);
      else
        g_GameOfLifeGrid->dead(i,j);
    }
        else{
      if(g_GameOfLifeGrid->getNumOfNeighbors(i,j) == 3)
        g_GameOfLifeGrid->live(i,j);
      else
        g_GameOfLifeGrid->dead(i,j);
    }
      }
    }
    g_GameOfLifeGrid->next();
    g_GameOfLifeGrid->decGen();
  }
```

그림1. Serial하게 실행되는 code로 구현된 Game of Life코드

   next() : pthread를 사용하지 않고 main함수에서 serial하게 game of life를 구현할 때 사용하는 함수이다. m_Grid전체에 m_Temp전체의 값을 저장하는 기능을 한다.

```
void GameOfLifeGrid::next()
{
  for(int i = 0; i < m_Rows; i++)
    for(int j = 0; j < m_Cols; j++)
      m_Grid[i][j] = m_Temp[i][j];
}
```

그림2. next()함수의 코드

이를 parallelize하게 game of life를 구현하려면 thread들에게 grid의 구간을 나눠 수행하게 하여 구현할 수 있다. 이 때 memory의 locality를 높이기 위해 나누는 기준은 Grid를 row에 대해 nprocs개의 구간으로 나누도록 했고 각각의 thread는 자신의 구간의 row 범위를 pthread_create에서 argument로 넘겨받은 구조체 내의 int형 멤버인 from, to를 통해 알게 된다. 그 다음 각각의 thread는 한 generation에서 m_Temp에 from~to범위의 row의 cell들의 값을 계산한 뒤 다음 generation으로 넘어가기 전에 next(const int from, const int to)함수를 호출한다. 이 때 thread들 간의 동기화는 pthread_barrier_t를 이용해 해결한다. Thread를 이용하여 실행될 때 프로그램은 barrier를 count에 생성할 thread의 개수를 넣어 init시키고 각각의 thread는 next함수를 호출하기 전과 next함수를 호출하면서 decGen을 통해 generation을 감소시킨 다음 다음 generation으로 넘어가기 전에 pthread_barrier_wait를 하여 thread간의 진행과정이 parellel하게 진행되도록 한다.

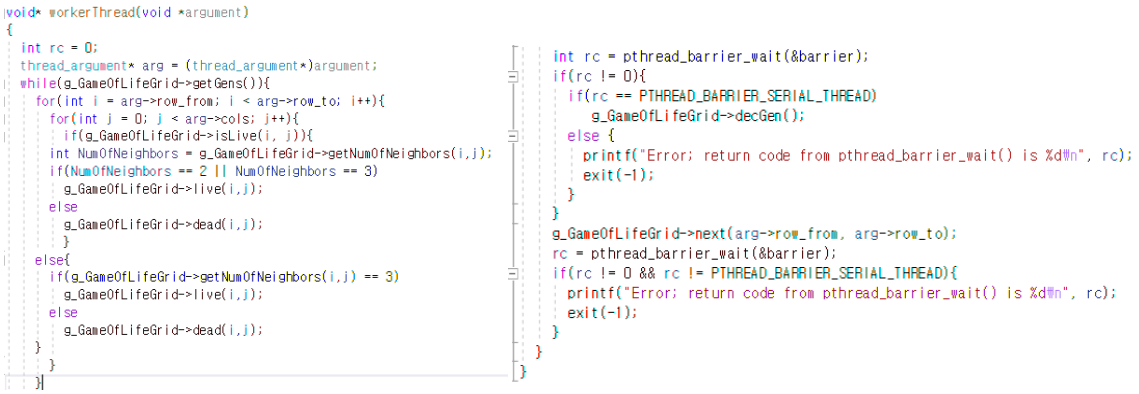

그림3. multiple threads에서 thread들이 실행하는 workerThread의 코드

next(const int from, const int to) : pthread를 생성해서 사용하여 game of life를 구현할 때 pthread가 수행하는 함수인 workerThread함수 내부에서 사용하는 함수이다. m_Grid의 from~to범위의 row의 셀에 m_Temp의 from~to범위의 row의 셀의 값들을 저장하는 기능을 한다.

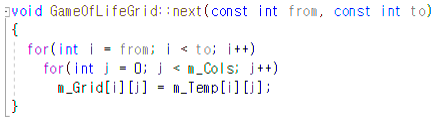

그림4. next(const int from, const int to)함수의 코드

2. Make a performance comparison by increasing the number of threads

row = 10, col = 10, gen = 20 환경에서 실행시간

nprocs	0	1	3	5	10
실행시간	0.000157	0.000427	0.000836	0.001162	0.002031

Grid의 크기가 작을 땐 serial 버전에서 가장 빨랐으며 thread가 늘어날수록 오히려 느려지는 결과를 얻었다.

row = 100, col = 100, gen = 100 환경에서 실행시간

nprocs	0	1	3	5	10	15	20
실행시간	0.047793	0.050388	0.021847	0.01676	0.013548	0.017968	0.031323

Grid의 크기를 어느정도 늘리자 thread를 10개 가량까지 늘릴 때 serial버전보다 약 네배 가량 빠른 performance를 보였으며 더 늘렸을 땐 다시 performance가 악화됐다.

row = 1000, col = 1000, gen = 100 환경에서 실행시간

nprocs	0	1	3	5	10	15	20
실행시간	3.05748	3.1028	1.13803	0.704242	0.39252	0.336051	0.045538

Grid의 크기를 1000*1000으로 매우 크게 늘리자 thread를 10개까지 늘려 나갔을 때 Serial에서의 실행시간을 nprocs로 나눈 값에 가깝게 실행시간이 줄어들었고 15개까지는 실행시간이 조금 더 줄어들었으며 20개에선 실행시간이 다시 늘어나는 모습을 보였다.