집중교육2- Ptheard Game of Life (3차과제 보고서)

201521032 한태희

1. 디자인과 구현

3번째 실습은 Game of Life (약칭 GOL)을 순차적인 c 프로그래밍과 병렬적인 POSIX Thread (약칭 Pthread) 프로그래밍을 이용해 2가지 버전으로 구현하고, 여러가지 변인들을 바꾸어가며 2가지 알고리즘의 성능을 테스트하는 과정을 진행했다.

GOL의 셀 업데이트는 Gen(주기) 단위로 진행되고, 주기가 넘어갈 때 모든 셀이 동시에 업데이트 되는 규칙이다. 이를 순차적인 방식으로 구현하기 위해선 다음 주기를 나타내는 임시 그리드에다가 입력하고, 그 다음에 전체 계산이 끝났으면 임시 그리드와 현재 주기를 나타내는 데이터 그리드의 역할을 스왑하는 방식을 사용한다.

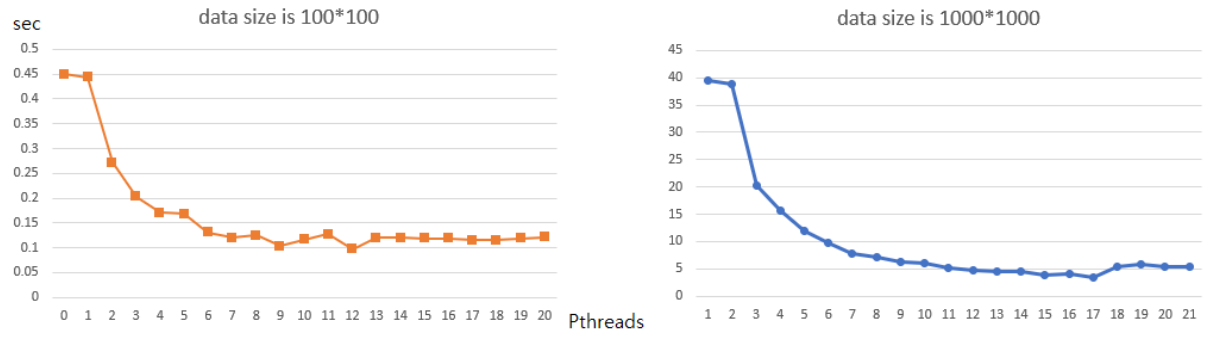
병렬적으로 GOL을 분할하기 위해선 이 규칙에서 종속성을 분석해야 한다. 임시 그리드의 (i, j) 번째 인덱스는 (i-1, j-1), ... (i+1, i+1)의 8개의 데이터 그리드에서 값을 읽기만 하고, 병렬적인 연산 과정에서 동시에 같은 주소의 임시 그리드에 접근할 필요도 없다. 그러므로 임시 그리드의 각 셀은 한 주기 내에선 아무런 데이터 종속성이 없고, 모든 셀을 한 주기 내에선 병렬적으로 연산해도 상관이 없다. 그리고 현재 주기를 나타내는 데이터 그리드는 읽히는 역할만 하고 스왑 이외엔 값이 변경되지 않으므로 뮤텍스를 걸어줄 필요도 없다.

위와 같은 분석을 통해서 Pthread를 이용해 GOL을 병렬로 처리하는 프로그램을 작성했다. 메인 함수는 (그리드 크기)/(스레드의 개수) 크기만큼 각 스레드들에게 처리해야 할 구간을 분배한다. (Blocked assignment) 이때, 나머지 k가 생길 경우 0번 스레드부터 k-1번째 스레드까지 할당 크기를 1씩 늘려서 전체 범위를 커버할 수 있도록 한다.

각 Pthread 인스턴스는 main 스레드에서 분담받은 from부터 to까지 순차적인 연산과 동일한 연산을 진행한다. 그리고 barrier를 통해 다른 스레드들의 주기 연산 종료를 기다린다. 주기 연산이 완전히 종료되면 0번 스레드는 스왑 작업을 진행하고, 나머지 스레드들은 0번 스레드를 기다린다. 0번 스레드의 스왑 작업이 완료되면 다음 주기 연산을 반복하게 된다.

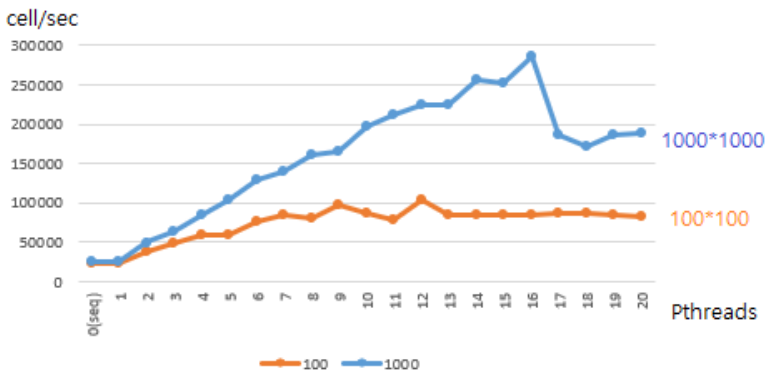
2. 성능 측정과 분석 – gridSize, Pthread 개수의 변화

구현한 GOL 프로그램을 이용해서 여러 조건일 때 프로그램의 작동 성능을 분석했다. 작동 성능은 시간 측정 함수를 이용해 구현한 함수가 시작할 때부터 종료될 때 까지의 **초 단위 시간**을 측정했다. 입력한 독립 변수는 **Pthread의 개수** (0은 순차적 실행을 의미), **데이터 그리드의 크기** 100by100 혹은 1000by1000)이다. Gen (주기)는 888회를 실행하는 것으로 고정하였다.



두 데이터의 크기일 때 모두 Pthread 개수가 많아질수록 성능이 향상되었지만, 어느 지점부터 병렬 처리로는 속도의 향상을 기대할 수 없는 지점에 도달한다. 더 이상 줄일 수 없는 시간은 각 주기당 순차적 스왑 작업 같은 serial한 연산으로 인한 시간에 가까워져 간다고 볼 수 있다.

데이터의 크기가 다른 두 경우는 위의 그래프만으로는 객관적인 평가가 어렵다. 더 객관적인 평가를 위해서, cell/sec이라는 단위를 만들어 그래프를 다시 그렸다. 이는 전체 프로그램이 1초에 몇 개의 cell을 처리하였는지를 속도로 나타내는 지표이다.



cell/sec 단위로 프로그램의 성능을 평가하면, 같은 pthread 개수를 이용해도 데이터 사이즈가 큰 경우가 더 빠른 처리량을 얻을 수 있음을 알 수 있다. 100*100 사이즈에서는 7개 초과의 Pthread의 개수는 유의미한 성능 향상을 보이지 못했고, 1000*1000 사이즈에서는 16개 초과의 Pthread 개수는 유의미한 성능 향상을 보이지 못했다.

이렇게 데이터 사이즈가 작은 경우는 Pthread가 조금만 늘어나도 금방 성능 한계에 도달하고 마는데, 이 이유는 작업이 너무 잘게 분할되어 순차적 실행 보다 병렬로 작업을 분해하고 관리하는 오버헤드가 더 크기 때문일 것이다. 따라서, GOL 같은 완전 병렬 실행이 가능한 어플리케이션도 아키텍처의 환경에 맞게 적당하게 작업을 분할하는 것이 오히려 성능 향상에 도움이 될 것이다.