Journal of The Institute of Electronics and Information Engineers Vol.53, NO.6, June 2016 http://dx.doi.org/10.5573/ieie.2016.53.6.080
ISSN 2287-5026(Print) / ISSN 2288-159X(Online)

논문 2016-53-6-9

그래프 컬러링과 OpenMP를 이용한 병렬 메쉬 스무딩 알고리즘의 성능 분석

(Performance Analysis of a Parallel Mesh Smoothing Algorithm using Graph Coloring and OpenMP)

신 명 규*, 김 지 범**

(Myeonggyu Shin and Jibum Kim©)

요 약

본 논문에서는 그래프 컬러링과 OpenMP를 사용한 병렬 메쉬 스무딩 알고리즘을 제안하고 공유메모리 기반의 슈퍼컴퓨터를 이용하여 제안하는 병렬 메쉬 스무딩 알고리즘의 성능 분석을 수행하였다. 제안하는 병렬 메쉬 스무딩 알고리즘은 그래프 컬러링 방법을 통해 전체 메쉬를 여러 개의 독립적인 집합 (색깔)으로 나눈 후 각각의 독립적인 집합에 대하여 OpenMP 라이브러리를 사용하여 순차적으로 병렬 메쉬 스무딩을 수행하는 방법이다. 실험을 통하여 여러 가지 그래프 컬러링 방법과 색깔 순서 재배열 방법이 병렬 메쉬 스무딩의 효율성에 미치는 영향에 대해서 알아보았다. 또한, OpenMP의 루프 스케줄링 방법이 병렬 메쉬 스무딩의 효율성에 끼치는 영향에 대해서 알아보았다.

Abstract

We propose a parallel mesh smoothing algorithm using graph coloring and OpenMP library for shared memory many core computer architectures. The proposed algorithm partitions a mesh into independent sets and performs a parallel mesh smoothing using OpenMP library. We study the effect of using various graph coloring and color reordering algorithms on the efficiency of performing the proposed parallel mesh smoothing algorithm. We also investigate the influence of using various OpenMP loop scheduling methods on the parallel mesh smoothing efficiency.

Keywords : 메쉬 최적화, 병렬 컴퓨팅, OpenMP, 그래프 컬러링

Ⅰ. 서 론

메쉬 질은 미분방정식과 관련된 수치 시뮬레이션 혹은 컴퓨터 그래픽스와 관련된 시뮬레이션에서 시뮬레이션 해의 정확도와 효율성 모두에 큰 영향을 미친다고 알려져 있다[1]. 따라서 메쉬를 이용하는 시뮬레이션에서는 좋은 질의 메쉬를 사용하는 것이 중요하다. 최근에는 자동으로 다양한 형태의 메쉬를 생성하는 여러 가지 상용 소프트웨어와 공개 소프트웨어 들이 널리 쓰이고 있다. 이러한 소프트웨어들을 사용하여 메쉬를 생성할 경우 모델링하는 물체나 영역에 따라서 안 좋은 질의 메쉬가 생성되거나 inverted element를 포함하는 메쉬가 생성되는 경우가 종종 생기게 된다[2, 14]. Inverted element란 주어진 요소 (element)의 넓이가 음인 요소를 의미 하는데 이러한 inverted element는 수치 시뮬레이션 시 잘못된 해를 생성한다. 따라서 메쉬를 생성한 후 invertedelement가 있다면 그러한 inverted

* 학생회원, 인천대학교 컴퓨터 공학부
(Incheon National University)
** 정회원, 인천대학교 컴퓨터 공학부
(Incheon National University)
© Corresponding Author (E-mail : jibumkim@inu.ac.kr)
※ 이 논문은 인천대학교 2015년도 자체 연구비 지원에 의하여 연구되었음.
Received ; February 2, 2016 Revised ; May 18, 2016
Accepted ; May 26, 2016

element를 제거해야 한다. 또한, 생성된 메쉬의 질을 높이기 위해서 시뮬레이션에 사용하기 전에 메쉬 스무딩을 수행하여 메쉬 질을 높이게 된다.

수치 시뮬레이션을 수행 시 보다 정교하게 물체를 모델링하고 정확한 해를 얻기 위해서는 크고 정교한 메쉬를 필요로 한다. 하지만, 기존의 컴퓨팅 환경에서 메쉬 스무딩을 수행하면 메쉬 크기가 커질 경우 메쉬 스무딩의 수행시간이 매우 오래 걸린다는 단점이 있었다. 이러한 문제를 해결하고자 최근에는 OpenMP, MPI, CUDA 등의 병렬 컴퓨팅 기술을 메쉬 스무딩에 사용하는 연구가 수행되고 있다. 기존에 병렬 메쉬 스무딩에 관한 연구를 살펴보면, Shankar[2]는 분산 메모리 기반의 슈퍼컴퓨터에서 메쉬에 있는 가장 안 좋은 질을 갖는 요소의 메쉬 질을 향상 시킬 수 있는 방법인 log-barrier 알고리즘을 제안하였다. Gorman[3]은 OpenMP와 MPI를 동시에 사용할 수 있는 hybrid OpenMP/MPI를 사용한 병렬 메쉬 스무딩 알고리즘을 제안하였다. Jiao[4]는 메쉬의 코너와 같은 메쉬 안의 중요한 특징들을 보존할 수 있는 병렬 메쉬 스무딩 알고리즘을 제안하였다. 본 논문과 가장 관련 있는 연구들은 Freitag[5]과 Benitez[6]의 연구가 있다. Freitag 는 공유 메모리 기반의 슈퍼 컴퓨터와 분산 메모리 기반의 슈퍼 컴퓨터에서 각각 수행 가능한 병렬 메쉬 스무딩 알고리즘을 제안하였다[5]. 하지만, 공유 메모리 슈퍼컴퓨터에 대해서는 이론적인 결과만 보여주었고 분산 메모리 슈퍼컴퓨터를 사용하여 병렬 메쉬 스무딩을 수행한 결과를 보여주었다. Benitez는 공유메모리 기반의 슈퍼컴퓨터에서 병렬적으로 inverted element를 제거하는 메쉬 언텐글링 (untangling) 방법을 수행하면서 동시에 메쉬 질을 높이는 알고리즘을 제안하였다[6].

본 논문에서는 그래프 컬러링 (graph coloring)과 OpenMP 라이브러리를 이용한 병렬 메쉬 스무딩 알고리즘을 제안하고 공유 메모리 기반의 슈퍼 컴퓨터에서 성능 분석을 수행하고자 한다. 제안하는 방법은 주어진 메쉬에 대해서 inverted element의 생성을 방지하고자 그래프 컬러링을 이용해 전체 메쉬를 여러 개의 독립적인 집합 (independent set)으로 분할한 후 각각의 독립적인 집합에 대하여 순차적으로 병렬 메쉬 스무딩을 수행하는 방법이다. 다양한 성능 분석을 통하여 공유 메모리 기반의 컴퓨터에서 여러 가지 그래프 컬러링 방법과 색깔 순서 재배열 방법이 병렬 메쉬 스무딩에 끼치는 영향에 대해서 알아본다. 또한, OpenMP 루프 스케줄링이 효율성에 미치는 영향에 대해서 조사 한다.

Ⅱ. 본 론

1. 메쉬 스무딩 (mesh smoothing)

메쉬 스무딩 방법 중 가장 간단하고 널리 쓰이는 방법은 라플라시안 스무딩 (Laplacian smoothing) 방법이다. 라플라시안 스무딩 방법은 메쉬 스무딩 시 메쉬 안의 점의 다음 위치를 그 점의 이웃 한 점들 위치의 평균으로 이동하는 방법이다. 라플라시안 스무딩 방법은 계산이 간단하다는 장점 때문에 널리 쓰이지만 메쉬 질을 보장하지 못한다는 단점이 있고 종종 inverted element를 생성한다는 단점이 있다[5]. 이를 극복하고자 제안된 방법이 최적화에 기반을 둔 메쉬 스무딩 방법이다. 이 방법은 메쉬 질을 측정할 수 있는 메쉬 질 측정 함수를 설정하고 이를 최적화 한다. [1]을 보면 메쉬 질을 측정할 수 있는 다양한 메쉬 질 측정 함수를 소개하고 있다. 본 논문에서는 메쉬 질 측정 함수로 널리 쓰이는 메쉬 안의 삼각형 요소 (element)의 모양을 측정하는 함수를 사용하였다[1, 5]. 이 메쉬 질 측정 함수는 이상적인 삼각형의 모양이 정삼각형이다. 메쉬 안의 i번째 삼각형 요소의 메쉬 질 (q_i)은 식 (1)과 같이 측정 된다.

$$q_i = \frac{\sum_{j=1}^{3} l_j^2}{4\sqrt{3}\,A} \tag{1}$$

여기서 A는 삼각형의 넓이를 의미하고 l_j는 삼각형의 j번째 변의 길이를 의미 한다. q_i 값은 이상적인 삼각형 (예: 정삼각형)일 경우 1의 값을 갖게 되고 이상적인 삼각형 모양에서 벗어날수록 큰 값을 갖게 된다.

본 논문에서는 메쉬 스무딩 시 흔히 쓰이는 지역 (local) 메쉬 스무딩 방법을 사용하였다[5, 14]. 지역 메쉬 스무딩이란 메쉬 안의 내부 점 한 개를 정해서 그 점과 이웃한 요소들을 이용하여 목적 함수를 정한 후 그 목적 함수를 최소화 하는 위치로 그 안쪽 점을 이동하는 방법이다. 자세한 지역 메쉬 스무딩에 관한 내용은[5]을 참고하면 된다. 본 논문에서는 이웃한 요소들의 메쉬 질의 합을 목적 함수로 설정하였다. 본 논문에서는 메쉬 스무딩시 메쉬의 바깥쪽 경계 (boundary)에 있는 점들은 움직이지 않고 모든 안쪽 점들에 대해서 메쉬 스무딩을 반복 수행한다. 메쉬 스무딩은 메쉬 질이 만족스럽게 좋아질 때까지 반복 수행하게 된다.

메쉬 안의 내부 점들을 이동시킬 때 주어진 목적함수를 최소화 시키면서 inverted element가 생기지 않는

위치로 점들을 이동 하게 되는데 본 논문에서는 steepest descent 알고리즘을 사용하여 최적화를 수행하였다.

2. 병렬 메쉬 스무딩

먼저 지역 메쉬 스무딩 알고리즘을 병렬화 할 때에 생기는 문제에 대해 알아보자. 그림 1은 (a) 최초의 메쉬, (b) 병렬 메쉬 스무딩 후의 메쉬를 보여주고 있다. 그림 1(b)에서와 같이 인접한 두 내부 점들 (v1, v2)이 각각 다른 코어에서 동시에 메쉬 스무딩이 이루어진다면 메쉬 안에 inverted element (회색 삼각형)가 생길 수 있다. 이러한 inverted element는 계산 시뮬레이션 시 잘못된 해를 생성하게 된다[2]. 이러한 inverted element의 생성을 방지하기 위해서 본 논문에서는 병렬 메쉬 스무딩을 수행할 경우 전체 메쉬를 그래프 컬러링을 이용하여 메쉬 안의 점들을 여러 개의 독립적인 집합 (independent set)으로 나누었다. 메쉬 안의 어떤 점들의 집합이 독립적인 집합이 되려면 그 집합에 속한 모든 점들은 서로 인접 (선으로 연결)하지 않아야 한다. 독립적인 집합에 속한 점들은 서로 인접하지 않으므로 동시에 병렬 메쉬 스무딩이 가능하고 inverted element가 생기는 문제가 발생하지 않는다.

본 논문에서 제안하는 병렬 메쉬 스무딩 방법은 다음과 같다. 먼저 전체 메쉬를 다음 장에서 자세히 설명할 그래프 컬러링 방법을 통해 여러 개의 독립적인 집합 (색깔)으로 나눈 후 하나의 색깔에 속한 점들 (집합)에 대하여 OpenMP를 사용하여 동시에 병렬메쉬 스무딩을 수행한다. 하나의 색깔에 대하여 메쉬 스무딩이 끝나면 이를 다른 색깔들에 대하여 반복 수행한다. 모든 색깔에 대하여 메쉬 스무딩이 끝나면 이를 전체 메쉬 질이 만족스러울 정도로 좋아질 때까지 반복 수행 한다.

3. 그래프 컬러링

주어진 메쉬를 여러 개의 독립적인 집합으로 분할하기 위하여 본 논문에서는 세 가지의 서로 다른 그래프 컬러링 방법을 사용하였다. 첫 번째 방법은 Jones and Plassmann (JP)이 제안한 그래프 컬러링 방법이다[11]. JP 그래프 컬러링 방법은 병렬로 수행 가능하도록 제안되었지만 본 논문에서는 JP 그래프 컬러링 방법의 시리얼 (serial) 버전을 사용하였다. JP 그래프 컬러링 방법은 먼저 그래프 안의 모든 점들에 대하여 난수를 할당한 후 각 점에서 이웃한 점들과 비교하여 할당받은

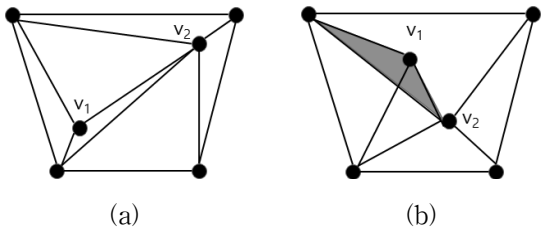

그림 1. (a) 최초의 메쉬 (b) 병렬 메쉬 스무딩 후의 메쉬
Fig. 1. (a) Initial mesh (b) Output mesh after parallel mesh smoothing.

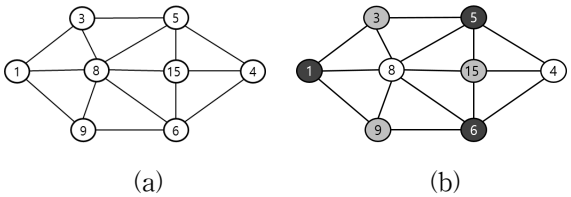

그림 2. (a) 최초의 그래프 (b) JP 그래프 컬러링 방법으로 컬러링한 결과
Fig. 2. (a) Initial graph (b) Output graph after performing JP graph coloring.

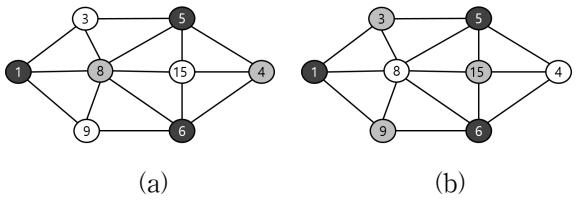

그림 3. (a) LDF 그래프 컬러링 방법으로 컬러링한 결과 (b) SDL 그래프 컬러링 방법으로 컬러링한 결과
Fig. 3. (a) Output graph after performing LDF graph coloring (b) Output graph after performing SDL graph coloring.

난수 값이 가장 큰 값 (local maximum)인 경우 그 점에 대하여 가능한 가장 낮은 색깔 (0번 색깔) 부터 할당 한다. 이 단계가 끝나면 색깔이 칠해져 있지 않은 점들에 대하여 색깔이 칠해지지 않은 이웃한 점들 중 local maximum인 점을 찾고 그 점에 가능한 가장 낮은 색깔 부터 반복해서 할당 한다. 그림 2 에서는 (a) 최초의 그래프 (메쉬)와 (b) JP 방법으로 그래프 컬러링을 수행한 결과의 예를 보여주고 있다. 점 (vertex)안의 숫자들은 할당받은 난수이다. 그래프 컬러링 결과 총 3개의 색깔이 사용되었고 0번, 1번, 2번 색깔은 그림에서 각각 회색, 흰색, 검정색 이다.

두 번째 그래프 컬러링 방법은 largest degree first (LDF) 그래프 컬러링 방법이다[7]. LDF 그래프 컬러링 방법은 JP 그래프 컬러링 방법과 상당히 유사하지만 local maximum을 정할 때 난수를 이용하는 것이 아니

표 1. 천둥 슈퍼컴퓨터의 계산 노드 사양
Table1. Specification of chundoong super computer.

CPU	2 × Intel Xeon E5-2560 (Sandy Bridge-EP, 8-core 2.00GHz)
메모리	128 GB (DDR3 1,600MHz)
운영 체제	Red Hat Enterprise Linux 6.3

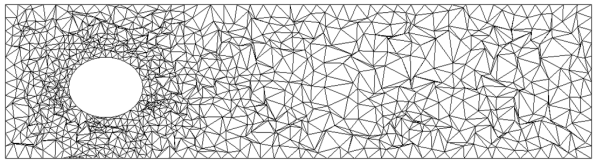

그림 4. 2D 실린더 메쉬
Fig. 4. 2D Cylinder mesh.

라 각 점의 차수 (degree) 값을 사용한다. LDF 방법에서도 난수를 부여 하지만 그 값을 충돌 회피 목적으로 사용한다. 즉, 두 점이 같은 차수 값을 갖는 경우에 더 큰 난수 값을 갖는 점에게 가능한 가장 낮은 색깔을 할당함으로써 충돌을 회피 한다. 그림 3 (a) 에서는 그림 2 (a)의 최초 그래프를 LDF 그래프 컬러링 방법을 사용하여 그래프 컬러링을 수행한 결과를 보여주고 있다. 점안의 숫자들은 할당받는 난수이다.

세 번째 방법은 smallest degree last (SDL) 그래프 컬러링 방법이다[7]. SDL 그래프 컬러링 방법은 크게 두 개의 단계로 구성되어 있다. 첫 번째 단계는 가중치를 할당하는 단계로서 먼저 메쉬 안에 있는 점 들 중 가장 차수가 작은 모든 점들을 찾고 그 점들에 가능한 가장 작은 가중치를 할당 한다. 가중치가 할당된 점들을 그래프에서 제외 하고 남아 있는 점들에 대하여 가중치를 반복하여 할당 한다. 이를 모든 점에 가중치가 할당될 때 까지 반복 수행하게 된다. 두 번째 단계는 LDF 그래프 컬러링 방법과 상당히 유사하다. LDF 그래프 컬러링 방법과의 차이는 색깔 할당 시 각 점들의 차수가 아닌 각 점에 있는 가중치를 사용하여 가중치가 가장 높은 점들부터 색깔을 할당하게 된다. 이 경우에도 난수는 LDF 그래프 컬러링 방법처럼 충돌 회피를 위해 사용 된다. 그림 3 (b)에서는 SDL 그래프 컬러링 방법으로 그래프 컬러링을 수행한 예를 보여 준다. 그림 2 (a)의 그래프에 대해서 그래프 컬러링을 수행한 결과 세 가지 그래프 컬러링 방법 모두 세 개의 색깔을 사용하여 그래프 컬러링을 수행할 수 있었고 JP와 SDL그래프 컬러링 방법은 동일한 결과를 보여주었다.

4. 실험 환경

병렬 메쉬 스무딩 실험을 위하여 서울대학교에 있는 천둥 슈퍼 컴퓨터를 사용하였다[8]. 실험에는 천둥 슈퍼 컴퓨터에서 8개의 CPU 2개가 연결되어 총 16개의 코어가 있고 메모리가 2개로 나뉘어져 불균일 기억 장치 접근 (NUMA) 구조를 가진 공유 메모리 기반의 계산 노드를 할당 받아 실험하였다[12]. 천둥 슈퍼 컴퓨터의 계산 노드에 대한 자세한 사양은 표 1에 나와 있다. 실험에는 그림 4에 있는 2D 실린더 메쉬를 사용하였다. 이 메쉬는 총 1,196개의 점과 2,240개의 요소를 가지고 있고 식 (1)로 메쉬 질을 측정한 결과 평균적인 메쉬 질은 2.08, 최악의 메쉬 질은 65.9로 측정되었다.
여러 개의 다른 2D 메쉬를 사용하여 실험하였지만 비슷한 실험 결과 패턴을 갖는 것을 확인하였고 이 논문에서는2D 실린더 메쉬를 사용하여 실험한 결과를 분석하였다. 병렬 메쉬 스무딩은 MSTK 라이브러리를 이용하여 구현하였다[13]. 병렬 컴퓨팅을 위한 OpenMP 라이브러리는 OpenMP 버전 3.0을 사용하였다[9].

4.1 그래프 컬러링 방법이 메쉬 스무딩 시간에 미치는 영향

먼저 실험에 사용된 실린더 메쉬에 대하여 그래프 컬러링 자체에 걸리는 시간을 측정하였다. 세 가지 그래프 컬러링 방법 모두에 대해서 0.01초 이내로 그래프 컬러링이 완료되었다. 전체 수행시간을 그래프 컬러링 시간과 병렬 메시 스무딩 수행 시간의 합이라고 하면 그래프 컬러링 시간은 전체 수행 시간에서 차지하는 비중이 1% 이하로 전체 수행 시간에 미치는 영향은 매우 작음을 알 수 있었다. 메쉬 질에 관하여서는 어떠한 그래프 컬러링 방법을 선택하더라도 최종 메쉬의 질 차이는 크게 없었고 모두 평균 메쉬 질이 대략 1.0인 메쉬로 수렴하였다.

두 번째로 세 가지 그래프 컬러링 방법 중 어떠한 그래프 컬러링 방법이 병렬 메쉬 스무딩 시 가장 수행시간이 빠른 (효율적) 방법인지 알아내고자 하였다. 실험에 사용된 메쉬의 경우 그래프 컬러링을 수행하는데 JP 그래프 컬러링 방법은 7개의 색깔, LDF와 SDL 그래프 컬러링 방법은 각각 6개의 색깔이 사용 되었다. 그래프 컬러링 시 난수를 사용하기 때문에 그래프 컬러링 결과가 매번 일치하지는 않았지만 대개의 경우 JP 그래프 컬러링 방법이 그래프 컬러링 시 다른 그래프 컬러링 방법에 비해서 더 많은 색깔을 사용하였다.

그림 5에서는 코어 수 변화와 그래프 컬러링 방법의 변화에 따른 병렬 메쉬 스무딩 수행 시간이 나와 있다. 6개의 색깔을 사용한 LDF와 SDL 그래프 컬러링 방법

이 7개의 색깔을 사용한 JP 그래프 컬러링 방법에 비해서 코어 16개를 사용한 경우 수행시간이 최대 76%까지 감소하는 것을 확인할 수 있었다. LDF와 SDL 그래프 컬러링 방법은 비슷한 성능을 보였지만 전반적으로 가장 빠른 수행시간을 보이는 것은 SDL 그래프 컬러링 방법이었다.

이 실험을 통해 LDF와 SDL 그래프 컬러링 방법과 같이 메쉬를 독립적인 집합으로 만드는데 필요한 색깔의 개수가 적을수록 병렬 메쉬 스무딩 수행 시간이 빠른 것을 확인할 수 있었다. 메쉬 안의 점들을 적은 수의 색깔을 사용하여 그래프 컬러링을 수행 하면 많은 수의 색깔을 사용할 때 보다 한 색깔에는 더 많은 수의 점들이 포함되어 있게 된다. 이러한 경우 한 번에 많은 수의 점들이 동시에 병렬로 스무딩될 수 있으므로 병렬 메쉬 스무딩의 수행시간이 훨씬 단축 되는 것을 확인할 수 있었다. 이러한 실험 결과를 바탕으로 이 실험 이후에는 세 가지 그래프 컬러링 방법 중 전반적인 수행 시간이 가장 빠른 그래프 컬러링 방법인 SDL 그래프 컬러링 방법을 사용하여 그래프 컬러링을 수행하였고 다른 실험들을 수행하였다.

그림 5의 실험 결과를 보면 모든 그래프 컬러링 방법에서 코어수를 8개에서 16개로 증가시켰을 때 수행 시간 단축이 거의 이루어지지 않아 병렬 효율 (parallel efficiency)의 향상이 거의 없음을 확인할 수 있었다. 그 주된 이유는 크게 두 가지로 분석할 수 있었다. 첫 번째는 코어를 16개로 모두 사용 시 천둥 컴퓨터의 계산 노드의 메모리 구조는 NUMA 구조이기 때문에 코어를 8개만 사용할 때에 비하여 캐시 미스율이 크게 증가하는 것을 확인할 수 있었다. 계산 노드에서 16개의 코어를 모두 사용한 경우 같은 지역에 위치한 메모리 (local memory)가 아닌 원격에 위치한 메모리 (remote memory)에서 데이터를 읽는 경우가 많아졌고 캐시 미스 (주로, L2 캐시 미스)로 인하여 메쉬 스무딩 수행시간이 크게 증가하였다. 코어 16개일 때 병렬 효율이 떨어지는 또 다른 이유는 16개의 코어를 사용 시 OpenMP 쓰레드를 스케줄링 할 때 생기는 로드 밸런싱 (load balancing) 오버헤드 로 인하여 수행 시간이 증가하는 것을 확인 할 수 있었다.

4.2 색깔 순서 재배열 이 메쉬 스무딩 시간에 미치는 영향

병렬 메쉬 스무딩을 위하여 주어진 메쉬를 여러 개의 색깔로 나누게 되면 어떤 색깔에 속한 점들부터 병렬

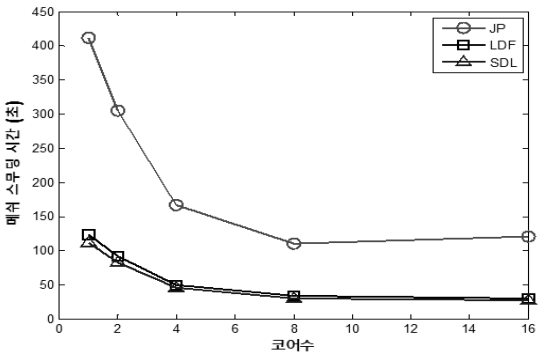

그림 5. 코어 수 변화에 따른 여러 가지 그래프 컬러링 방법의 메쉬 스무딩 수행 시간 (초)

Fig. 5. Mesh smoothing time for various graph coloring methods in terms of the number of cores.

메쉬 스무딩을 수행할지 정해야 한다. 이를 색깔 순서 재배열 (color reordering)이라 한다. 시리얼 (serial)한 컴퓨팅 환경의 경우 메쉬 스무딩의 효율을 높이기 위하여 메쉬 스무딩시 수행하는 점 순서 재배열 (vertex reordering)에 관한 연구는 몇몇 진행된 것이 있지만 [10], 병렬 메쉬 스무딩시 고려해야 하는 색깔 순서 재배열에 관한 연구는 기존에 연구된 논문이 없다.

본 논문에서 사용된 그래프 컬러링 방법들의 경우 그래프 컬러링을 수행 시 사용되는 전체 색깔의 수를 줄이기 위하여 대개의 경우 낮은 색깔 (0번에 가까운 색깔)에 더 많은 수의 점들이 속하게 되고 반대로 가장 큰 색깔에는 적은 수의 점들만 속하게 되는 것을 확인할 수 있었다. 본 논문에서는 총 네 가지의 색깔 순서 재배열 방법을 제안하였는데 이를 두 가지 기준으로 비교하였다. 첫 번째로 전반적으로 메쉬 질이 좋은 색깔을 먼저 메쉬 스무딩 (good quality first)을 할지 전반적으로 메쉬 질이 나쁜 색깔을 먼저 메쉬 스무딩 (poor quality first) 하는 것이 효율성이 높은지 비교하였다. 두 번째로는 한 색깔에 속한 점의 개수가 많은 색깔부터 메쉬 스무딩 (max vertices first)을 하는 게 효율성이 높은지 반대로 색깔에 속한 점의 개수가 적은 색깔부터 메쉬 스무딩 (min vertices first)을 하는 것이 효율성이 높은지 서로 비교하였다.

그림 6에서는 위에서 언급한 네 가지의 색깔 순서 재배열 방법을 비교한 결과가 나와 있다. 실험 결과 색깔 순서 재배열은 메쉬 스무딩 시간에 큰 영향을 미치는 것으로 파악되었다. 어떤 방법을 선택하는 지에 따라 최대 6.5배 정도 메쉬 스무딩 시간이 더 걸렸다. 네 가지 방법 중 전반적으로 가장 좋은 성능을 보인 것은 여러 개의 색깔 중 전반적으로 메쉬 질이 좋지 않은 색깔

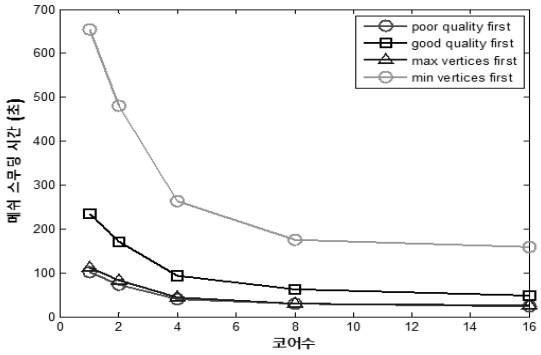

그림 6. 코어 수 변화에 따른 여러 가지 색깔 순서 재배열 방법의 메쉬 스무딩 수행 시간 (초)
Fig. 6. Mesh smoothing time for various color reordering methods in terms of the number of cores (sec).

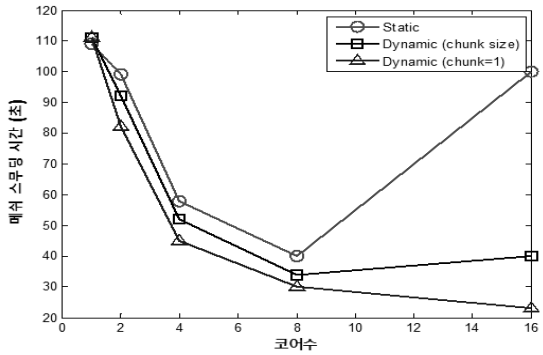

그림 7. 코어 수 변화에 따른 여러 가지 OpenMP 루프 스케줄링 방법의 메쉬 스무딩 수행 시간 (초)
Fig. 7. Mesh smoothing time for various OpenMP loop scheduling methods in terms of the number of cores (sec).

부터 순서대로 병렬 메쉬 스무딩을 수행하는 poor quality first 방법이었다. 전반적으로 메쉬 질이 좋지 않은 색깔의 경우 그 색깔에 속한 점들이 최적의 점 위치로부터 많이 벗어나 있는 경우들이 대부분이었고 이런 점들부터 메쉬 스무딩을 수행해야 전반적인 메쉬 스무딩 수행 시간이 감소하는 것을 확인할 수 있었다.

또한 전반적으로 한 색깔 안에 있는 점의 개수가 많은 색깔부터 스무딩을 수행하는 것이 수행시간이 짧은 것을 확인할 수 있었다. 동시에 더 많은 수의 점을 병렬로 스무딩해서 점들을 빠르게 최적의 위치로 이동 시키는 것이 점을 동시에 더 적은 수의 점을 병렬로 스무딩하는 것보다 더 효율적이라는 것을 발견할 수 있었다.

4.3 OpenMP 루프 (loop) 스케줄링이 메쉬 스무딩 시간에 미치는 영향

병렬 메쉬 스무딩에서 OpenMP 루프 스케줄링 시 어떠한 OpenMP 루프 스케줄링 방법이 병렬 메쉬 스무딩 시 가장 효율적인지 확인하기 위한 실험을 수행하였다. OpenMP 루프 스케줄링의 기본 스케줄링 방법인 정적 (static) 스케줄링 방법과 각 쓰레드에게 덩어리 (chunk) 개수만큼 작업을 할당하는 동적 (dynamic) 스케줄링 방법을 비교하는 실험을 수행하였다. OpenMP에서의 기본적인 정적 스케줄링 방법은 전체 N번의 반복횟수가 있고 T개의 쓰레드가 있을 경우 각 쓰레드에게 N/T개의 작업이 할당 되는 방식이다. 본 논문에서는 동적 스케줄링을 사용할 때에는 덩어리 크기 (chunk size)로 컬러에 속한 점 개수/쓰레드 개수를 정하는 방법과 덩어리 크기로 항상 1을 주는 방법을 비교해 보았다.

그림 7에서는 여러 가지 OpenMP 루프 스케줄링을 서로 비교 실험한 결과를 보여 준다. 실험 결과 OpenMP 루프 스케줄링 방법 중에 동적 스케줄링 방법을 사용하고 덩어리 크기를 1로 정하는(즉, 점 하나만을 할당) #pragma omp parallel for schedule(dynamic, 1)을 사용하는 것이 가장 효율적임을 확인할 수 있었다. 앞의 실험 결과에서 보듯이 NUMA 구조로 인한 영향과 OpenMP에서의 쓰레드 스케줄링에 의한 오버헤드와 지연으로 인하여 코어가 8개에서 16개로 증가할 때 메쉬 스무딩 시간의 변화가 크지 않고 몇몇 경우 오히려 메쉬 스무딩 시간이 증가함을 확인할 수 있었다.

OpenMP 루프 스케줄링 시 덩어리 크기를 1로 정하는 동적 스케줄링 방법이 성능이 가장 좋은 원인을 분석해 보면 메쉬 질이 좋지 않은 메쉬의 경우 메쉬 안의 점들은 메쉬 질의 차이가 많이 나는 경우가 대부분 이었다. 이러한 경우 각 점 별로 메쉬 스무딩을 수행하는 시간이 많이 차이가 나기 때문에 OpenMP 루프 스케줄링시 덩어리 크기를 크게 하지 않고 1로 정하여 메쉬 스무딩이 끝난 점들을 빠르게 다른 쓰레드에게 할당 시켜주는 것이 보다 효율적인 루프 스케줄링 방법임을 알 수 있었다.

Ⅲ. 결 론

본 논문에서는 그래프 컬러링과 OpenMP를 이용한 병렬 메쉬 스무딩 알고리즘을 제안하였다. 제안하는 병렬 메쉬 스무딩 알고리즘을 공유 메모리 기반의 천둥 슈퍼 컴퓨터를 사용하여 다양한 성능분석을 수행하였다. 실험 결과 코어 16개를 사용하였을 경우 병렬 메쉬 스무딩의 수행시간이 코어 1개를 사용하였을 때에 비하여 최대 76%까지 감소하였다. 계산 노드 안의 코어 16개를

모두 사용하였을 경우 NUMA 구조로 인한 캐시 미스율의 증가와 OpenMP 쓰레드 스케줄링에 의한 오버헤드로 인하여 병렬 효율이 떨어지는 것을 발견할 수 있었다. 여러 가지 그래프 컬러링 방법의 효율성을 비교한 결과 그래프 컬러링 방법 중 가장 적은 수의 색깔을 사용하여 메쉬를 독립적인 집합으로 분할한 그래프 컬러링 방법이 가장 효율적임을 알 수 있었다.

여러 가지 색깔 순서 재배열 방법 중에는 가장 안 좋은 메쉬 질을 갖는 색깔부터 메쉬 스무딩을 수행하는 색깔 순서 재배열 방법이 가장 효율적인 것을 확인할 수 있었다. 가장 효율성이 떨어지는 색깔 순서 재배열 방법을 선택한 경우 가장 효율성이 좋은 색깔 순서 재배열 방법에 비해서 최대 6.5배 수행 시간이 오래 걸렸다. 공유 메모리 기반의 슈퍼 컴퓨터에서 OpenMP 라이브러리 사용 시 적절한 OpenMP 루프 스케줄링 방법이 성능에 큰 영향을 미침을 알 수 있었는데 가장 효율성이 좋았던 것은 동적 루프 스케줄링 방법을 사용하면서 덩어리 크기를 1로 놓는 방법이었다. 이는 실험에서 사용된 메쉬와 같이 메쉬 안의 점들의 메쉬 질 차이가 많이 발생할 때 효율적인 방법임을 확인할 수 있었다

다음 연구에서는 보다 많은 수의 코어를 사용할 수 있는 분산 메모리 기반의 슈퍼컴퓨터에서 메쉬 분할 방법들이 메쉬 스무딩의 효율성에 끼치는 영향에 대해서 분석해볼 계획이다.

REFERENCES

[1] J.R. Shewchuk., “What is a good linear element? Interpolation, conditioning, and quality measures,” in Proc. of the 11th International Meshing Roundtable, Sandia National Laboratories, pp. 115-126. 2002.

[2] S.P. Sastry and S.M. Shontz, “A parallel log-barrier method for mesh quality improvement and untangling,” Engineering with Computers, Vol. 30, no. 4, pp. 503−515, 2014.

[3] G. Gorman, J. Southern, P. Farrell, M. Piggott ,G. Rokos, P. ,Kelly, “Hybrid OpenMP/MPI anisotropic mesh smoothing. in Proc. of the international conference on computational science,” procedia computer science, vol 9. pp 1513 - 1522, 2012.

[4] X. Jiao and P.J. Alexader, “Parallel feature -preserving mesh smoothing, in Proc. of International Conference on Computational Science,” pp. 1180-1189, 2005.

[5] L. Freitag, J.M. Plassmann P, “A parallel algorithm for mesh smoothing,” SIAM J. Sci Comput. Vol. 20, no. 6, pp. 2023 - 2040, 1999.

[6] D. Benıtez, E. Rodrıguez, J.M. Escobar, R. Montenegro, “Performance evaluation of a parallel algorithm for simultaneous untangling and smoothing of tetrahedral meshes,” in Proc. of the 22nd international meshing roundtable. Springer International Publishing, pp. 579 - 598, 2013.

[7] J. R. Allwright, R. Bordawekar, P. D. Coddington, K. Dincer, and C. L. Martin, “A comparison of parallel graph coloring algorithms,” Technical report, SCCS-666, Northeast Parallel Architectures Center at Syracuse University, 1995.

[8] Chundoong Supercomputer, Center for Manycore Programming, Seoul National University, 2016.

[9] OpneMP Architecture Review Board, OpenMP API, http://www.openmp.org/.

[10] S.M. Shontz and P.M. Knupp, “The effect of vertex reordering on 2D local mesh optimization efficiency,” in Proc. of the 17th International Meshing Roundtable, Sandia National Laboratories, pp. 107-124, 2008.

[11] M.T. Jones and P.E. Plassmann, “A parallel graph coloring herustic,” SIAM Journal of Scientific Computing, Vol. 14, no. 654, 1993.

[12] J. Kim, “The Effect of Mesh Reordering on Laplacian Smoothing for Nonuniform Memory Access Architecture-based High Performance Computing Systems”, Journal of The Institute of Electronics and Information Engineers Vol. 51, NO. 3, March 2014.

[13] R.V. Garimella, “MSTK - A flexible infrastructure library for developing mesh based applications,” in Proc. of the 13rd International Meshing Roundtable, Sandia National Laboratories, pp. 203-212. 2004.

[14] J. Kim, T. Panitanarak, S.M. Shontz, “A multiobjective mesh optimization framework for mesh quality improvement and mesh untangling,” International Journal for Numerical Methods in Engineering, Vol. 94, pp. 20-42, 2013.

저 자 소 개

신 명 규(학생회원)

2015년 인천대학교 컴퓨터공학부 학사

2015년～현재 인천대학교 컴퓨터공학과 석사과정

<주관심분야 : 계산과학, 고성능컴퓨팅>

김 지 범(정회원)

2003년 연세대학교 전기전자공학 학사

2005년 연세대학교 전기전자공학 석사

2012년 미국 펜실베니아 주립대학 컴퓨터 공학 박사

2013년 Los Alamos National Lab, 박사후 연구원

2013년～현재 인천대학교 컴퓨터공학부 조교수

<주관심분야 : 계산과학, 고성능 컴퓨팅>