2014년 3월 전자공학회 논문지 제 51권 제 3 호
Journal of The Institute of Electronics and Information Engineers Vol. 51, NO. 3, March 2014 http://dx.doi.org/10.5573/ieie.2014.51.3.082

논문 2014-51-3-9

NUMA구조를 가진 고성능 컴퓨팅 시스템에서의 메쉬 재배열의 라플라시안 스무딩에 대한 효과

(The Effect of Mesh Reordering on Laplacian Smoothing for Nonuniform Memory Access Architecture-based High Performance Computing Systems)

김 지 범*

(Jbium Kim©)

요 약

우리는 고성능 컴퓨팅 (high performance computing) 시스템에서 메쉬 재배열이 라플라시안 스무딩에 대해서 어떠한 효과가 있는지 연구한다. 구체적으로, 우리는 nonuniform memory access (NUMA) 구조의 고성능 컴퓨팅 시스템에서 Reverse-Cuthill Mckee 알고리즘을 사용하여 메쉬를 재배열하고 메쉬질을 높이기 위하여 라플라시안 스무딩을 사용한다. 먼저 하나의 코어를 사용하여 메쉬 재배열의 라플라시안 스무딩에 대한 속도 향상을 조사한 후에 NUMA구조의 멀티코어 시스템에서 OpenMP를 이용하여 병렬화할 경우 메쉬 재배열의 라플라시안 스무딩에 대한 속도 향상에 대하여 조사한다.

Abstract

We study the effect of mesh reordering on Laplacian smoothing for parallel high performance computing systems. Specifically, we use the Reverse-Cuthill McKee algorithm to reorder meshes and use Laplacian Smoothing to improve the mesh quality on Nonuniform memory access architecture-based parallel high performance computing systems. First, we investigate the effect of using mesh reordering on Laplacian smoothing for a single core system and extend the idea to NUMA-based high performance computing systems.

Keywords : High performance computing, Laplacian smoothing, mesh reordering, OpenMP, NUMA

Ⅰ. 서 론

최근에 하드웨어의 급속한 발달과 함께 멀티코어를 이용한 고성능 컴퓨팅 (high performance computing) 시스템에 대한 관심이 증가하고 있다. 고성능 컴퓨팅이란 슈퍼 컴퓨터와 같이 멀티 코어 시스템을 갖춘 시스템을 이용하여 계산 속도를 획기적으로 높이는 것을 의미 한다. 많은 고성능 컴퓨팅 시스템 중에 이 논문에서는 최근에 많이 사용되는 불균일 기억 장치 접근 (non-uniform memory access: NUMA) 구조 (architecture)의 고성능 컴퓨팅 시스템을 고려하고자 한다. NUMA 메모리 구조는 기존의 대칭형 다중 처리 (SMP: symmetric multiprocessing) 와 같은 UMA (Uniform Memory Access) 구조 의 문제점을 해결하고자 등장한 구조이다[1~2]. SMP 구조에서는 모든 메모리 접근이 공유되는 특성을 갖기 때문에 코어 개수가 증가할수록 모든 코어가 하나의 메모리를 공유함으로써 지

* 정회원, 인천대학교 컴퓨터공학부
(Department of Computer Science and Engineering, Incheon National University)
© Corresponding Author(E-mail: jibumkim@incheon.ac.kr)
※ 이 논문은 미래 창조과학부가 지원한 2013년 산업융합원천기술개발사업 (No. 10041332)의 지원을 받아 수행되었음.
접수일자: 2014년1월6일, 수정완료일: 2014년2월25일

연 시간이 커지고 캐쉬 미스 (cache miss)가 많이 생기는 문제가 있었다. 이를 해결하고자 NUMA 구조의 시스템이 등장하게 되었다. NUMA 구조의 시스템에서는 각각의 프로세서에 별도의 메모리를 제공함으로써 SMP 구조가 갖는 문제를 해결할 수 있고 SMP 구조에 비해서 확장성이 좋다는 장점이 있다. 따라서 현재 공유 메모리 시스템 (shared memory system) 구조의 고성능 컴퓨팅 시스템에서는 NUMA 구조의 시스템이 많이 사용되고 있다[1~2].

메쉬 (mesh)는 물체와 같은 어떠한 기하학적인 도메인 (geometric domain) 을 이산화 (discretize)한 결과물을 의미한다. 주로, 기계공학, 수치해석, 컴퓨터 그래픽스 분야에서 널리 쓰인다. 메쉬는 물체를 점 (vertex), 선 (edge), 면 (face), 영역 (region) 등으로 이루어진 배열로 기하학적인 도메인을 이산화한 결과물이다. 메쉬 재배열이란 다양한 알고리즘을 이용해 점, 선, 면 등의 순서를 바꾸는 것을 의미 한다. 우리는 이 논문에서 Reverse Cuthill-McKee (RCM) 알고리즘을 써서 메쉬를 재배열 하고자 한다[3]. RCM 알고리즘은 구현이 용이하고 메쉬 데이터의 지역성 (data locality)을 높일 수 있다는 장점이 있다. 이 논문에서는 RCM 알고리즘을 이용한 메쉬 재배열이 수치 알고리즘의 속도 향상에 어떠한 영향을 미치는지 고성능 컴퓨팅 시스템에서 조사하고자 한다. 구체적으로 수치 알고리즘 중에서 기계공학, 수치해석, 컴퓨터 그래픽스 분야 등에서 널리 사용되는 라플라시안 스무딩 (Laplacian smoothing) 방법에 대해서 고려하고자 한다[4]. 라플라시안 스무딩이란 주로 메쉬의 질 (mesh quality)을 높이는데 널리 사용되는 방법 중의 한가지로써 새로운 점의 위치를 현재 점의 이웃들의 평균으로 구하는 방법이다.

구현상으로는 수치 알고리즘을 고성능 컴퓨팅 시스템에서 라플라시안 스무딩의 병렬화를 위해서 OpenMP 라이브러리를[5] 사용하였다. OpenMP는 공유 메모리 (shared memory) 고성능 컴퓨팅 시스템에서 많이 사용되는 병렬 프로그래밍 언어이다. 실험을 통해서 RCM 방법을 이용한 메쉬 재배열이 라플라시안 스무딩을 수행 할 때 얼마나 계산 속도의 향상이 있는지 또한 코어 수가 늘어남에 따라서 NUMA 구조에서 얼마만큼의 계산 속도 향상이 있는지 알아보고자 한다.

Ⅱ. Non-uniform Memory Access (NUMA) 구조의 고성능 컴퓨팅 시스템

1. NUMA 멀티코어 시스템의 구조

NUMA 컴퓨터 구조는 그림 1과 같이 각각의 프로세서 (소켓)에 별도의 독립적인 메모리를 할당한다. 각각의 코어가 메모리에 접근 할 경우에는 그 위치에 따라서 지역 메모리 접근 (local memory access)와 원격 메모리 접근 (remote memory access)으로 나뉜다. 각각의 코어는 지역 메모리와 원격 메모리 모두에 접근할 수 있다. 한 프로세서가 다른 프로세서와는 cross-chip interconnect를 통해서 연결되어 있다. 최근 논문에 따르면 cross-chip interconnect를 통한 원격 메모리 접근은 지역 메모리 접근에 비해서 1.5배에서 2배정도 메모리 접근 시간이 걸리는 것으로 알려져 있다[1~2].

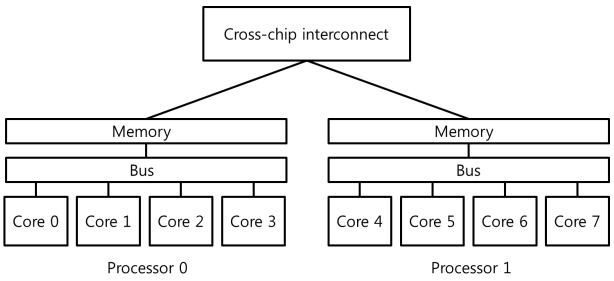

그림 1. 2개의 프로세서를 가진 NUMA-멀티코어 구조 시스템의 예
Fig. 1. Example of NUMA-multicore systems with 2 processors.

2. 관련된 연구

NUMA 시스템에서의 성능 향상 방법, 스케줄링 방법, 메모리 할당 방법 등에 대해서 주로 운영체제와 관련된 부분에서 많은 연구가 이루어져왔다[9~11]. [9]에서는 NUMA 시스템에서 프로그램을 수행하는데 성능 저하가 일어나는 원인들에 대한 다양한 분석이 이루어졌다. 주된 성능 감소 원인으로는 빈번한 원격 메모리 접근 시도로 분석되었다. [10]에서는 NUMA 시스템에서 dynamic 스케줄링 방법을 통해서 데이터의 지역성을 높이는 방법이 연구 되었다. [11]에서는 NUMA 시스템의 구조를 이용하여 메모리를 할당하고 load balancing을 하는 방법들이 제안 되었다.

메쉬 재배열에 관한 연구는 주로 메쉬 재배열이 메쉬

최적화의 속도 향상에 어떠한 영향을 미치는지 싱글 코어에 대해서 이루어져 왔다[6~7]. 메쉬 재배열을 통해서 메쉬 최적화시의 최적화 속도 향상을 얻을 수 있었다[6~7]. 하지만, 이러한 메쉬 재배열 방법이 NUMA 구조의 멀티 코어 시스템에서는 어떠한 영향을 미치는지는 연구된 적이 없다.

Ⅲ. 메쉬 재배열 (Mesh reordering)

1. 메쉬

메쉬 (mesh)란 어떠한 기하학적인 도메인 (geometric domain)을 이산화 (discretize)한 것을 의미한다. 메쉬는 주로 배열의 형태로 나타내어지며 점 (vertex), 선 (edge), 면 (face), 영역 (region) 등으로 표현되어 질 수 있다.

2. 메쉬 재배열

메쉬는 배열의 형태로 나타내어지며 각 점, 선, 면, 영역에는 순서를 나타내는 숫자가 할당되게 된다. 메쉬 재배열이란 각각의 점, 선, 면, 영역에 할당된 숫자를 재배열하는 것을 의미 한다. 우리는 많은 메쉬 재배열 방법 중에 Reverse Cuthill McKee (RCM) 방법을 사용하고자 한다[3]. RCM 방법은 간단하면서도 쉽게 메쉬 재배열을 할 수 있다는 장점이 있다. RCM은 이전까지 주로 희소 행렬 (sparse matrix)의 대역폭 (bandwidth)을 줄이는데 사용되어 왔다. 하지만, 우리는 RCM 방법을 메쉬 재배열에 사용함으로써 메쉬 데이터의 지역성 (data locality)를 높이고자 한다. 데이터의 지역성을 높임으로 인해서 메모리 접근의 효율성을 높이고 캐쉬 미스율을 낮추고자 한다. RCM 방법은 최초로 시작하는 점 (root vertex)을 정하고 그 점으로부터 너비우선탐색 (breadth-first search) 방법으로 다음 순서의 점, 선, 면, 영역을 찾는다. 그림 2에서는 간단한 메쉬에서 RCM 방법을 사용하기 전후의 메쉬 순서를 보여준다. RCM방법을 사용해서 재배열을 한 후의 메쉬를 보면 인접한 점과 면이 비슷한 지역성을 갖고 있음을 알 수 있다.

Ⅳ. 라플라시안 스무딩 (Laplacian smoothing)

메쉬의 질은 메쉬 요소의 모양 (element shape) 을 통해서 주로 측정할 수 있다. 메쉬의 질은 편미분 방정식을 이용한 수치문제의해와 컴퓨터 그래픽스 응용분야에 큰 영향을 준다고 알려져 있다[8]. 삼각형 메쉬를 사용할 경우 요소의 모양이 주로 정삼각형에 가까울 경우 메쉬 질이 높다고 알려져 있다[8]. 메쉬의 질을 높이는 가장 간단하고 널리 쓰이는 방법 중의 하나는 라플라시안 스무딩 방법을 이용하는 것이다. 라플라시안 스무딩

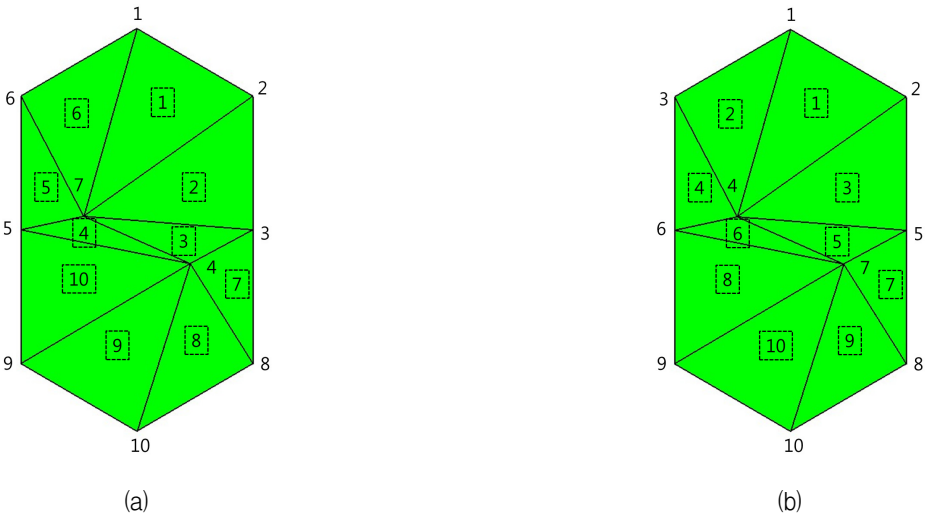

그림 2. (a) 최초의 메쉬 순서 (b) RCM 방법을 이용해 재배열된 메쉬의 순서. 이 그림에서 점선으로 둘러싸이지 않은 숫자는 점의 순서, 점선으로 둘러싸인 숫자는 면 의 순서를 각각 의미한다.

Fig. 2. (a) Initial mesh ordering (b) Updated mesh ordering using the RCM method.

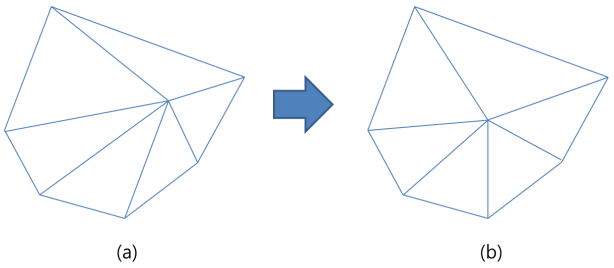

그림 3. (a) 라플라시안 스무딩 이전 메쉬 (b) 라플라시안 스무딩 이후의 메쉬
Fig. 3. (a) Initial mesh (b) Output mesh after performing Laplacian smoothing.

이란 메쉬에 있는 각 점에서, 새로운 위치를 그 점과 이웃된 점들을 이용하여 메쉬의 질을 높이는 방법이다. 어떠한 점이 주어져 있고 그 주위에 이웃한 점의 개수가 N개라고 하자. 그 점에 이웃한 점 중 i번째 점을 (p_i) 라고 한다면 라플라시안 스무딩을 이용한 새로운 점의 위치, $\overline{p_i}$, 는

$$\overline{p_i} = \frac{1}{N}\sum_{i=1}^{N} p_i$$

으로 표현할 수 있다. 그림 3에서는 라플라시안 스무딩의 한 예를 보여준다. 라플라시안 스무딩을 수행한 후 삼각형 모양이 정삼각형에 보다 가까워지는 것을 확인할 수 있다.

V. 실 험

우리는 실험을 통해 가장 먼저 RCM 방법을 통한 메쉬의 재배열이 주어진 메쉬에 대해서 라플라시안 스무딩을 행할 때 얼마만큼의 속도 향상이 있는지 알아보고자 한다. 두 번째로 멀티코어 상황에서 OpenMP를 사용하여 라플라시안 스무딩을 병렬화 했을 때 재배열이 NUMA 구조의 고성능 컴퓨팅 시스템에서 얼마나 속도 향상에 효과가 있는지 알아보고자 한다.

1. 실험 환경

실험에는 3개의 다른 종류의 메쉬를 사용하여 메쉬 질을 높이기 위한 라플라시안 스무딩을 수행하였다. 실험에 사용된 메쉬는 그림 4에서 보듯이 (a) 경사 메쉬, (b) 큐브 메쉬, (c) 센디아 메쉬가 사용되었다. 실험에는

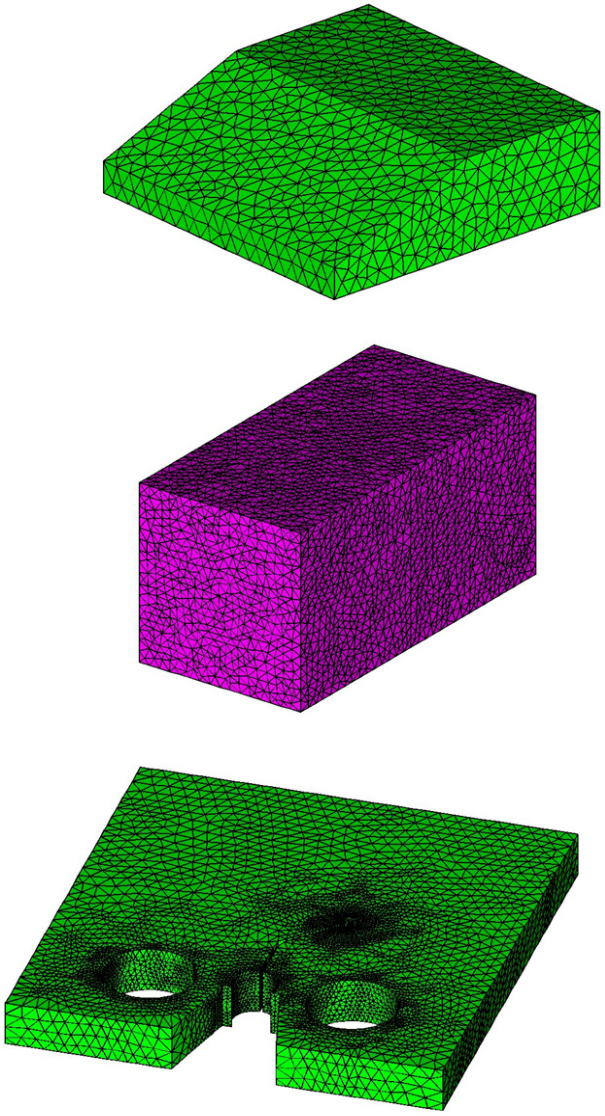

그림 4. 실험에 사용된 세가지 종류의 메쉬. (위) 경사 메쉬 (중간) 큐브 메쉬 (아래) 샌디아 메쉬
Fig. 4. Three example meshes. (Top) Slope mesh (Middle) Cube mesh (Bottom) Sandia mesh.

메쉬질의 향상을 위해서 라플라시안 스무딩을 약 1000회 반복하여 실험하였다. 메쉬를 읽고 쓰는데 MSTK 라이브러리를 사용하였다[12]. 라플라시안 스무딩의 병렬화를 위해서 OpenMP 버전 3.1이 사용되었다[5]. 실험에 사용된 HPC 컴퓨터는 다음과 같다. 총 16개의 코어가 존재하고 4개의 소켓 (프로세서)를 가지고 각각의 소켓은 4개의 코어 (quad core)를 가지는 고성능 컴퓨터 이다. 각 프로세서는 Intel Xeon CPU를 탑재하고 2.80 GHz의 속도를 가진다. 각 프로세서의 캐쉬 크기는

6,144 KB이다. 메쉬질에 관하여서는 라플라시안 스무딩을 행한 1000회 반복해 실험한 결과 재배열을 수행한 메쉬와 수행하지 않은 메쉬 사이의 질 차이는 없었다.

2. 메쉬 재배열의 라플라시안 스무딩의 속도 향상

가장 먼저 OpenMP를 사용하여 라플라시안 스무딩 방법을 병렬화 하지 않고 하나의 코어만 사용했을 경우 NUMA 구조에서 메쉬 재배열에 의한 라플라시안 스무딩의 속도 향상을 테스트 해보았다. 그림 5에서는 메쉬 재배열을 수행하기 전과 후의 라플라시안 스무딩의 수행 속도 결과가 나와 있다. 메쉬 재배열을 했을 경우 라플라시안 스무딩의 수행시간이 대략 20%정도까지 빨라짐을 알 수 있다. 코어를 하나만 사용할 경우 데이터의 지역성이 향상되어서 수행 시간이 감소됨을 실험을 통해서 알 수 있었다.

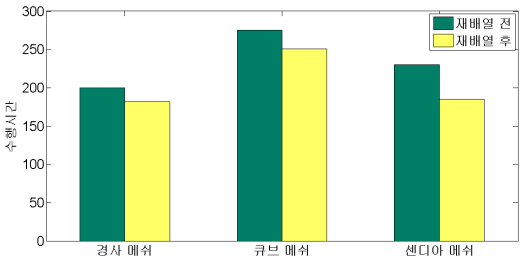

그림 5. 라플라시안 스무딩을 OpenMP를 사용하여 병렬화를 하기 전의 메쉬 재배열 전후의 라플라시안 스무딩 수행 시간

Fig. 5. Execution time comparison between the initial mesh and reordered mesh without using OpenMP library.

3. 멀티코어 환경에서 메쉬 재배열의 라플라시안 스무딩의 속도 향상

이번 실험에서는 OpenMP를 사용하여 라플라시안 스무딩을 병렬화 했을 경우 메쉬를 재배열 전과 재배열 한 후의 속도 향상을 측정해보고자 한다. NUMA구조의 멀티 코어일 경우 코어를 한 개 이상 사용하였을 때 의 속도 향상 결과가 세 가지 메쉬에 대하여 그림 6, 7, 8에 나와 있다. 우리가 처음 발견한 점은 재배열의 효과는 코어가 1개에서 4개까지 일 때 크다는 점이다. 코어 수를 1개에서 4개까지 늘렸을 때 메쉬 재배열 후 69%까지 라플라시안 스무딩의 수행시간을 줄일 수 있었다. 코어를 8개까지 사용한 경우 하나의 코어를 사용할 때

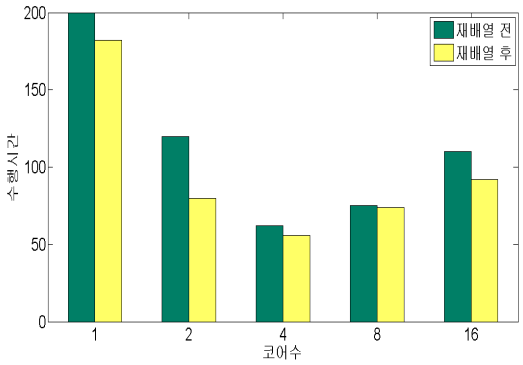

그림 6. 경사메쉬에서 코어수 증가에 따란 재배열 전후의 라플라시안 스무딩 수행 시간

Fig. 6. (Slope mesh) Execution time comparison between the initial mesh and reordered mesh with using OpenMP library.

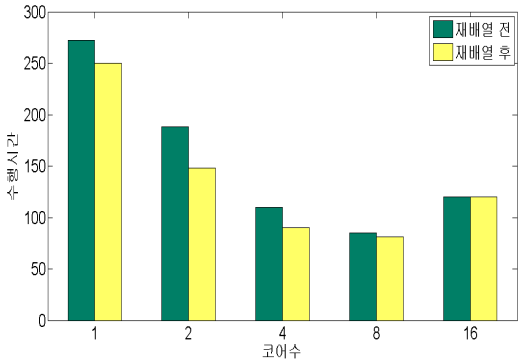

그림 7. 큐브메쉬에서 코어수 증가에 따란 재배열 전후의 라플라시안 스무딩 수행 시간

Fig. 7. (Cube mesh) Execution time comparison between the initial mesh and reorderd mesh with using OpenMP library.

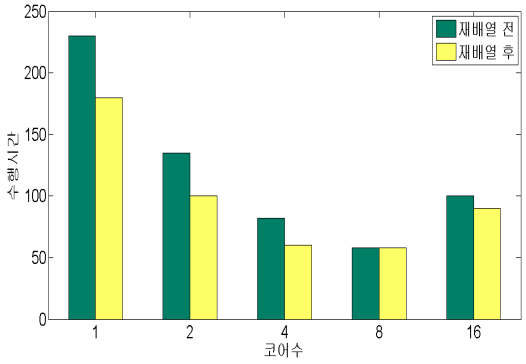

그림 8. 샌디아메쉬에서 코어수 증가에 따란 재배열 전후의 라플라시안 스무딩 수행 시간

Fig. 8. (Sandia mesh) Execution time comparison between the initial mesh and reordered mesh with using OpenMP library.

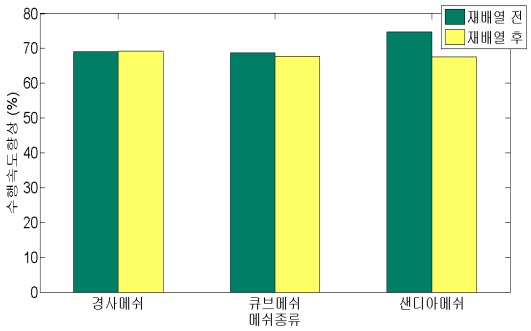

그림 9. OpenMP를 사용하여 라플라시안 스무딩을 병렬화를 하였을 경우 최대 수행속도 향상

Fig. 9. Maximum speed up when Laplacian smoothing is parallelized using OpenMP library.

보다 라플라시안 스무딩 수행시간을 75% (샌디아 메쉬의 경우)까지 줄일 수 있었다.우리가 발견한 다른 점은 코어수를 늘려도 항상 속도가 빨라지는 것은 아니라는 점이다. 코어가 1개에서 4개까지 증가할 경우에는 항상 속도가 향상되지만 코어가 4개에서 8개로 혹은 코어가 8개에서 16개로 증가시킬 경우에는 오히려 속도가 느려질 경우도 있었다. NUMA 시스템에서 코어 수 증가에 따른 성능 저하 원인은 여러 가지가 있을 수 있다. 이는 주로 NUMA 시스템의 메모리 구조로 인해 생기는 결과이다[2, 9]. NUMA구조의 경우 원격 메모리 접근 시간이 지역 메모리 접근 시간에 비해서 상당히 느리기 때문에 원격 메모리 접근시도가 많을 경우 성능 저하의 큰 원인이 된다. 다른 성능 저하 원인으로는 잦은 캐쉬 미스의 발생으로 생기는 속도 저하이다. 우리가 사용하는 16-core를 가진 HPC 시스템은 1개의 소켓이 4개의 코어를 가지고 있다. 따라서 16개의 코어를 사용할 경우에는 4개의 코어나 8개의 코어를 사용하는 것에 비해서 오히려 속도가 느릴 수도 있다[9]. 이는 [9]에서 보듯이 NUMA 구조의 멀티코어 시스템을 사용하는 경우 발생할 수 있는 문제 이다. 모든 코어를 사용하면 각 코어가 지역 메모리와 원격 메모리에 모두 접근을 시도하기 때문인데 앞에서 보듯이 cross-chip interconnect를 사용하여 다른 소켓에 있는 원격 메모리에 접근을 할 경우 1.5배에서 2배의 시간이 걸리기 때문에 코어수가 증가한다고 항상 수행속도가 빨라지는 것은 아님을 알 수 있다. 그림 9에서는 OpenMP 라이브러리를 사용하여 라플라시안 스무딩을 병렬화 하였을 경우 코어를 하나만 사용하여 라플라시안 스무딩을 수행하였을 경우와 비교했을 때 최대 수행 속도 향상의 결과가 그려져 있다. 여기에서 최대 수행 속도 향상은

$$\text{최대수행속도 향상} = \frac{\text{수행시간(병렬화전)} - \text{시간(병렬화후)}}{\text{수행시간(병렬화전)}}$$

으로 정의 하였다. 병렬화 후의 수행시간 (초)은 멀티코어를 사용하여 라플라시안 스무딩을 수행한 시간 중 (2개~16개 코어 중) 가장 빠른 수행시간을 사용하였다. OpenMP를 이용한 라플라시안 스무딩의 병렬화를 멀티코어에 적용함으로써 67%~75%의 라플라시안 스무딩의 수행 속도 향상을 달성 하였다. 재배열 전후에 최대 수행 속도 향상에는 큰 차이가 없었다.

VI. 결 론

우리는 이 논문에서 Reverse Cuthill McKee (RCM) 방법을 이용한 메쉬 재배열을 통해서 라플라시안 스무딩의 속도를 향상 시키는 방법을 제안하였다. 하나의 코어만을 사용하는 경우 간단한 RCM 방법을 통해서 20%까지 라플라시안 스무딩의 수행 시간을 줄일 수 있었다. NUMA 구조를 사용하는 16개의 코어를 가진 고성능 컴퓨팅 시스템에서는 RCM 방법으로 재배열된 메쉬에 대하여 OpenMP 라이브러리를 이용한 병렬화를 이용해서 67% - 75%까지 라플라시안 스무딩의 수행시간을 줄일 수 있었다. 하지만 모든 코어를 사용하는 경우 코어수를 늘렸음에도 불구하고 적은 코어를 사용할 때 보다 수행시간이 오히려 증가하는 경우가 자주 발생한다는 사실을 발견하였다. 그 이유는 NUMA 구조에서 모든 코어를 사용하는 경우 지연 시간이 큰 원격 메모리 접근을 자주 시도하기 때문이다. 이런 경우 코어수를 늘려도 오히려 수행 속도가 증가하는 경우가 발생함을 알 수 있었다. 이번 논문에서는 각 쓰레드가 프로세서로 자유롭게 할당되었다. 다음 연구에서는 OpenMP에서 쓰레드를 프로세서 별로 이동시키지 않는 프로그래밍을 하는 경우 수행시간에 어떠한 차이가 발생하는지 조사할 계획이다. 쓰레드가 프로세서 간 이동을 하지 않을 경우 보다 큰 속도 향상을 할 것으로 기대한다. 또한 다음 연구에서는 캐쉬 미스와 수행시간 속도 저하가 NUMA 시스템에서 어떠한 상관관계가 있는지 조사할 계획이다.

REFERENCES

[1] Z. Majo and T.R. Gross, Memory Management in NUMA Multicore Systems: Trapped between Cache Contention and Interconnect Overhead, Proc. of the International Symposium on Memory Management, pp. 11-20, 2011.

[2] S. Blagodurov and A. Fedorova. User-level Scheduling on NUMA Multicore Systems under Linux, Proc. of the 13 Annual Linux Symposium, pp. 81-92, 2011.

[3] J.A. George and J.W. Liu, Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, 1981.

[4] L. Freitag, On Combining Laplacian and Optimization-based Mesh Smoothing Techniques, Proc. of the 1997 Joint Summer Meeting of American Society of Mechanical Engineers (ASME) American Society of Civil Engineers (ASCE) and Society of Engineers Science (SES), pp. 37-44, 1997.

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

[6] S.M. Shontz and P.M. Knupp, The Effect of Vertex Reordering on 2D Local Mesh Optimization Efficiency, Proc. of the 17TH Internatioanl Meshing Roundtable, Sandia National Laboratories, pp. 107-124, 2008.

[7] H. Hoppe. Optimization of Mesh Locality for Transparent Vertex Caching, Proc. of the ACM SIGGRAPH, pp. 269-276, 1999.

[8] J. R. Shewchuk., What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures, Proc. of the 11th International Meshing Roundtable, Sandia National Laboratories, pp. 115-126. 2002.

[9] S. Blagodurv, S. Zhuravlev, and A. Fedorova. Contention-aware Scheduling on Multicore Systems, Proc. of the ACM SIGGRAPH, pp. 269-276, 1999.

[10] T. Ogasawara, NUMA-aware Memory Manager with Dominant-thread-based Copying GC, Proc. of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Language and Applications, pp. 77-390, 2009.

[11] L. Pilla, C. Ribeiro, D. Cordeiro, A. Bhatele, P. Navaux, J. Mehaut, and L. Kale, Improving Parallel System Performance with a NUMA-aware Load Balancer, INRIA-Illinois Joint Labarotory on Petascale Computing, Urbana, IL, Tech. Rep. TR-JLPC-11-02, vol, 20011, 2011.

[12] R.V. Garimella, MSTK - A Flexible Infrastructure Library for Develpoing Mesh Based Applications, Proc. of the 13rd International Meshing Roundtable, Sandia National Laboratories, pp. 203-212. 2004.

저 자 소 개

김 지 범(정회원)

2003년 연세대학교 전기전자공학 학사

2005년 연세대학교 전기전자공학 석사

2012년 미국 펜실베니아주립대학 컴퓨터공학 박사

2013년 Los Alamos National Lab, 박사후 연구원

2013년~현재 인천대학교 컴퓨터공학부 조교수

<주관심분야 : 계산과학, 고성능 컴퓨팅>