I/O 효율성을 위해 확장된 Multilevel 그래프 분할 기법

허준호° R. S. Ramakrishna
광주과학기술원 정보통신공학과
{jhher°, rsr}@kjist.ac.kr

Extended Multilevel Graph Partitioning Scheme for I/O Efficiency

Junho Her° R. S. Ramakrishna
Dept. of Info. and Comm., Kwangju Institute of Science and Technology

요 약

그래프 분할문제에서 대량의 그래프 데이터를 처리하는 것은 계산에서 걸리는 시간보다 파일 입출력을 수행하는 데 걸리는 시간의 비중이 크다. 본 논문은 수행 속도와 분할 성능에 있어서 우수한 그래프 분할 알고리즘 중 하나인 Multilevel Graph Partitioning에 대해 입출력 효율을 높일 수 있도록 확장하는 기법을 제안하고 그 구현에 대해 기술한다. 그래프를 컴퓨터의 가용 메모리를 기준으로 서브 그래프로 나누어 메모리 참조의 지역성이 향상되도록 기존의 Multilevel Graph Partitioning을 확장 하였다. 기존의 방식과 제안된 방식을 테스트 그래프들에 적용하여 그 수행시간을 비교한 결과 그래프 데이터의 크기가 컴퓨터의 주 메모리의 용량에 비해 어느 수준 이상으로 커지면서 제안된 알고리즘이 기존의 방식보다 수행시간에 있어서 좋은 결과를 보인다.

1. 서 론

그래프 분할 문제는 node와 edge들로 이루어진 그래프에 대해서 edge의 일부를 자름으로써 임의의 개수로 그래프를 나눌 때 잘려지는 edge의 개수가 최대한 적게 되는 해를 찾는 문제이다. 이와 같은 그래프 분할은 병렬 컴퓨팅에서 프로세서 간의 통신 비용을 줄이기 위한 태스크 스케줄링에 응용될 뿐만 아니라 VLSI 설계시의 레이아웃 문제나 여러 가지 과학 연산을 수행하는데 이용되고 있다[1].

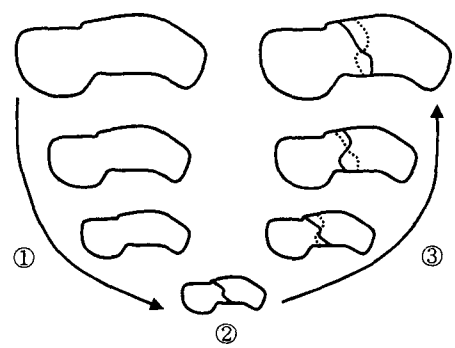
그래프 분할 알고리즘은 크게 spectral partitioning 알고리즘과 geometric 알고리즘 그리고 multilevel 알고리즘으로 나뉘게 되는데, 절대적으로 우위에 있는 알고리즘은 없다. 다만, multilevel 알고리즘이 분할 성능이나 수행 속도 면에서 다양한 문제들에 대해서 비교적 좋은 성능을 보이고 있다[1].

한편, 최근에 방대한 양의 데이터를 처리하는데 있어서 파일 입출력의 병목을 줄이는 것이 전체 수행 시간을 단축하는데 상대적으로 큰 효과가 있다는 것이 제기 되고 있다[2]. 실제로 데이터의 입출력을 운영체제의 메모리 정책에 전적으로 위임하지 않고 알고리즘 자체적으로 명시적인 입출력 스케줄링을 해주는 Out-of-Core 알고리즘 이 FFT (Fast Fourier Transform)에 적용되어 그 효과가 입증되었다[3]. 본 논문에서는 최근에 가장 주목 받는 그래프 분할 알고리즘인 multilevel 알고리즘을 방대한 그래프 데이터를 입출력 효과적으로 처리할 수 있도록 확장하는 기법을 제안한다. 큰 용량의 그래프 데이터를 컴퓨터의 가용 메모리를 기준으로 블록화하여 각 블록에 대해 입출력을 수행하고 명시적인 입출력을 행함으로써 파일 입출력의 빈도수를 줄여 전체 수행 시간을 줄일 수 있도록 하였다.

본 논문은 다음과 같이 구성된다. 2장에서는 multilevel graph partitioning 알고리즘에 대해서 살펴보고 이 알고리즘을 채택한 그래프 분할 라이브러리인 METIS에 대해서 간략히 소개한다. 3장에서는 기존의 multilevel 알고리즘을 대용량 그래프 데이터를 효율적으로 처리할 수 있도록 확장하는 기법에 대해 기술 하고, 4장에서는 제안된 기법으로 구현된 프로그램을 METIS의 프로그램(kmetis)과 비교하고 그 결과에 대해서 논한다. 마지막으로 5장에서는 결론을 맺는다.

2. Multilevel Graph Partitioning

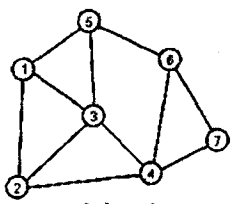
Multilevel algorithm은 기본적으로 세 단계를 거쳐서 그래프를 분할 하게 된다. 그림 1은 그 과정을 설명하고 있다. 첫째, 원래의 그래프를 그래프 매치(graph matching)를 통해서 그래프 압축(graph coarsening)을 큰 압축이 이루어 지지 않을 때까지 반복적으로 수행하고(①), 둘째 그 결과 그래프를 초기 분할한다(②), 마지막으로 다시 원래대로 그래프 확장(un-coarsening)을 반복적으로 수행 하면서 매 확장 단계마다 그래프 분할을 정제(partition refinement)하는 과정을 거치게 된다(③). 그림 1에서 점선으로 표시된 것은 정제된 분할을 뜻한다.



[그림 1] Multilevel Graph Partitioning

2.1 METIS 라이브러리

METIS는 George Karypis 등에 의해 구현된 그래프 분할 라이브러리로써, 앞서 기술한 multilevel 알고리즘을 이용하고 있는 대표적인 그래프 분할 라이브러리라고 할 수 있다[4][5]. 그림 2는 METIS에서 사용되고 있는 그래프 파일 포맷(b)과 그래프 자료 구조(c)의 예 보인다. 파일 포맷에서 첫 행은 그래프의 node 수와 edge 수를 의미한다. 이 예에서는 7개의 node와 11개의 edge로 이루어진 그래프를 표현한다. 다음의 각 행부터는 각 node 번호에 대하여 이웃하는 node의 번호를 순서대로 나타내는데, 이 예에서 2행은 node 1에 대한 이웃 node가 5, 3, 그리고 2임을 뜻 하고 이하 다른 node에 대해서도 마찬가지로 적용된다. 그래프 자료 구조에서 xadj은 각 node에 대해서 이웃 하는 node의 개수를 누적해서 나타내는 배열이고, adjncy는 파일 포맷에 있는 것과 같이 각 node에 대해서 이웃 하는 node 번호를 하나의 배열로 나타낸다.



(a) 예제 그래프

```
7 11
5 3 2
1 3 4
5 4 2 1
2 3 6 7
1 3 6
5 4 7
6 4
```

(b) 파일 포맷

```
xadj[ ] = {0,3,6,10,14,17,20,22}
adjncy[ ] = {2,3,5,1,3,4,1,4,5,2,3,6,7,1,3,6,4,5,7,4,6}
```

(c) 그래프 자료 구조

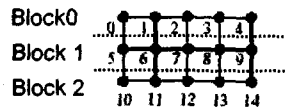
[그림 2] METIS의 그래프 표현 예

3. I/O 효율적인 기법

Multilevel 알고리즘에서 첫 번째 단계인 그래프 압축은 기본적으로 그래프 매칭을 수행하게 되는데 그래프 매칭이란 그래프의 독립적인 edge를 찾아 내는 것을 말하고, 이러한 매칭을 수행하기 위해서는 그래프 전체를 스캔(scan)하게 된다. 이런 계산은 그래프 자료가 커져 주 메모리 상에 다 올라올 수 없는 경우에는 과도한 I/O가 발생하여 전체 수행 시간을 영향을 미치게 된다. 그래프의 크기가 커질수록 수행 시간에 미치는 영향은 급진적으로 커지게 되므로, 본 논문에서는 그래프가 차지하는 용량이 주 메모리 보다 큰 경우 그래프를 나누어 각 서브 그래프에 대하여 그래프 매칭과 압축을 수행할 수 있도록 기존의 방법을 확장하는 기법을 제안한다. 이렇게 함으로써 매칭 및 압축 수행 시 과도한 페이지 폴트가 발생할 소지를 줄여서 전체 수행 시간을 개선 할 수 있도록 하였다. 특히 multilevel 알고리즘에서 데이터에 대해 전역적으로 참조를 요구하는 연산은 매칭 및 압축과정에서 있으므로 그림 1에서 압축과정(①) 중 그래프를 블록화하고 각 블록 그래프에 대해서 압축을 수행한 후 그래프를 합치고 합쳐진 그래프가 가용 메모리 보다 클 때는 계속해서 같은 과정을 반복하여 최종 그래프가 가용 메모리 보다 작을 때 기존의 방법대로 압축을 수행한다. 이에 대한 자세한 내용은 3.2절에서 다룬다.

3.1 그래프 블록화를 위한 자료구조

가용 메모리 보다 큰 그래프를 나누어 각 블록 그래프에 대하여 압축을 적용할 수 있기 위해서는 우선 각 블록 그래프의 연결관계가 유지되어야 하므로 경계부분 edge들을 따로 관리할 수 있는 bndadj1, bndadj2 배열을 추가하고 전체적인 블록의 분포를 distVtx 배열로 표현하도록 하였다. 그림 4는 이러한 자료 구조의 용례를 보이고 있다. 이 예에서는 그래프가 세 개의 블록으로 나뉘어 지는 경우를 보이고 있다.



(a) 그래프 예제 (세 블록으로 나뉘어지는 경우)

```
distVtx[ ]={0,5,10,15}
bndadj1[ ]={ 0, 1, 2, 3, 4, 5,  6, 7,  8, 9}
bndadj2[ ]={ 5, 6, 7, 8, 9,10,11,12,13,14}
```

(b) 블록 그래프 분포와 경계 edge를 위한 배열

```
Block 0
xadj[ ]={0,2,5,8,11,13}
adjncy[ ]={1,5,0,2,6,1,3,7,2,4,8,3,9}
Block 1
xadj[ ]={0,3,7,11,15,18}
adjncy[ ]={0,6,10,1,5,7,11,2,6,8,12,3,7,9,13,4,8,14}
Block 2
xadj[ ]={0,2,5,8,11,13}
adjncy[ ]={5,11,6,10,12,7,11,13,8,12,14,9,13}
```

(c) 블록화 된 그래프의 자료 구조

[그림 4] 그래프 블록화를 위한 자료구조의 용례

3.2 I/O 효율적인 Scheme

그림 5는 I/O 효율적인 그래프 압축과정에 대한 pseudo 코드이다. EstimateMem 함수는 node 수와 edge 수를 바탕으로 그래프에 쓰이는 자료 들을 byte로 계산해주는 루틴이다. EstimateMem는 전체 파일을 읽지 않고 첫 행만 읽고 예측하기 때문에 다소 부정확하지만 대략적인 블록화를 위한 것이므로 무리가 없다. MemAvail 함수는 가용 메모리를 보고해 주는 루틴으로 리눅스의 경우 /proc/meminfo 정보를 읽어서 FreeMem와 Cached 그리고 Buffers 항목의 값을 합산하여 리턴해 준다. Cache 메모리와 Buffer 메모리는 시스템의 상황에 따라 유동적으로 증감되는 메모리 항목으로 FreeMem 항목과 더불어 가용 메모리로 간주한다. 이 후 가용 메모리와 추정 메모리를 바탕으로 블록화 그래프 수(nblocks)를 결정한다. Coarsen$_{ic}$ 함수는

```
m ← EstimateMem (G0), M ← MemAvail(), nblocks ← m/M + 1
while (nblocks > 1) do {
  for k = 0 to nblocks-1 do {
      Gk ← read (fp, nvtxs)
      Gkc ( Coarsenic(Gk)
      write (Gkc)
  }
  for j=0 to nblocks-1 do {
    Gm ( append (Gm, Gj)
  }
  m ( EstimateMem (Gm), M ( MemAvail(),
  nblocks ← m/M + 1
}
```

[그림 5] I/O 효율적인 Scheme의 Pseudo 코드

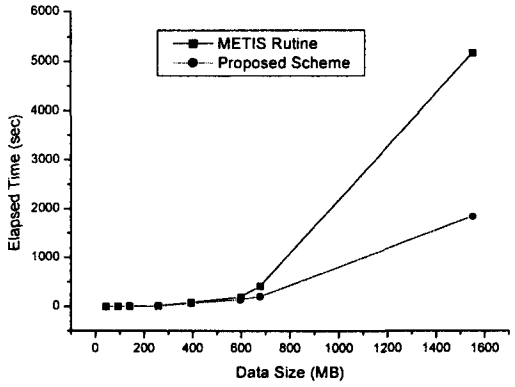
METIS의 그래프 압축 함수를 확장하는 것으로 압축을 마지막 단계에서 bndadj1과 bndadj2 그리고 distVtx를 업데이트 한다. 즉 압축과정에서 경계에 있는 노드에 변화가 생겼을 때 이를 반영하고 압축된 그래프에 대한 블록 분포를 재구성 한다. 각 블록에 대해 압축을 수행하고 마지막 단계에서 블록 그래프를 합친 후 그 그래프의 경우에도 역시 요구 메모리가 가용 메모리보다 크면 계속해서 반복됨을 보이고 있다. While 문을 벗어나게 되면 기존의 METIS의 압축 과정으로 넘어가게 된다.

4. 실험 및 고찰

실험은 단일 Linux PC (Pentium III 733Mhz CPU, 128MB Main Memory, kernel version: 2.2.18)에서 수행 되었으며 표 1의 그래프들은 sparse matrix로부터 기인된 그래프들이다. 요구 메모리는 전체 그래프 파일을 읽어서 예상 메모리를 산출하는 METIS의 한 유틸리티를 이용하여 측정하였다. 그러나 제안 기법에서 고안한 EstimateMem 함수가 리턴하는 값과 다소 차이가 있다. 기존의 METIS 프로그램인 kmetis를 이용하여 64-분할을 수행한 시간과 제안된 기법을 구현한 프로그램을 이용하여 측정한 64-분할을 수행한 시간을 비교하고 있다. 페이지 폴트 수는 C Shell 유틸리티인 ' time' 을 이용하여 kmetis의 수행 대해서만 측정한 것이다.

[표 1] 실험 Sparse Matrix에 따른 그래프 분할 (64분할)의 수행 시간 비교

Matrix	요구 메모리 (MB)	페이지 폴트 수	METIS 수행시간 (초)	제안기법 수행시간 (초)
BCSSTK31	42	1743	2.6	-
NASASRB	91	3893	5.54	-
TWOTONE	139	4182	9.16	12.04
LHR71	257	7425	14.92	15.32
PWTK	392	46682	80.28	65.83
PRE2	596	112864	190.20	136.90
MSDOOR	677	244980	423.98	203.45
LDOOR	1549	3072574	5170.57	1839.71



[그림 6] 제안된 기법과 kmetis의 수행 시간 비교

BCSSTK31과 NASASRB의 경우에는 요구 메모리가 가용 메모리보다 작기 때문에 제안된 기법에 적용되지 않고 바로 기존의 방법을 그대로 쓰게 되므로 수행시간은 기록하지 않았다. 그림 6은 수행시간을 그래프로 도식화하여 비교 하고 있다. 그림에서와 같이 LHR71(257MB)까지는 거의 비슷하거나 긴 수행 시간을 보이다가 PWTK(596MB)부터 제안 방법의 수행 시간이 짧아지기 시작하여 큰 용량의 그래프 데이터에 대해서 월등히 짧은 수행 시간을 보이고 있다. 즉 가용 메모리에 비해 요구 데이터가 클수록 입출력에 소모되는 시간이 급진적으로 증가하여 제안된 기법의 효과가 극명하게 드러나고 있다.

5. 결 론

본 논문에서는 주 메모리에 비해 큰 메모리를 요구하는 그래프 데이터에 대해 그래프의 블록화를 통해 I/O를 효율적으로 처리할 수 있는 확장된 Multilevel 그래프 분할 기법을 제안하였다. 이는 병렬화 기법에서 쓰이는 문제의 블록화와는 달리 입출력 효율성에 중점을 둔 기법으로 분산 메모리 형태의 병렬 처리에도 그대로 적용될 수 있다. 분산 메모리 형태의 병렬 처리 자체도 메모리의 한계를 어느 정도 보완 할 수 있으나 문제의 크기는 과학용 데이터의 경우 매우 클 수 있으므로 병렬화를 위해 나뉘어진 문제 자체도 지역 메모리의 한계를 넘어 서게 되므로 이러한 입출력 효율 증진을 위한 기법이 절실히 요구된다.

앞으로 병렬화 그래프 분할의 경우에 적합한 I/O 효율적인 기법을 고안하여 구현하고 그 효과를 고찰하는 것이 필요하다.

6. 참고 문헌

[1] A. Pothen, " Graph partitioning algorithms with applications to scientific computing" , In *Parallel Numerical Algorithms*, D. E. Keyes, A. H. Sameh and V. Venkatakrishnan (editors), Kluwer Academic Press, 1996.

[2] Sivan Toledo, " A survey of out-of-core algorithms in numerical linear algebra," In James M. Abello and Jeffrey Scott Vitter, editors, *External Memory Algorithms*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pp. 161-179, American Mathematical Society, 1999.

[3] Thomas H. Cormen, and David M. Nicol, " Performing Out-of-Core FFTs on Parallel Disk Systems," *Parallel Computing*, Vol. 24, pp. 5-20, Jan. 1998.

[4] G. Karypis and V. Kumar. " A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," SIAM J. Sci. Comput., Vol. 20, pp. 359-392, 1998.

[5] G. Karypis and V. Kumar. " METIS: Unstructured graph partitioning and sparse matrix ordering system," *Technical Report TR 97-061*, Department of Computer Science, University of Minnesota, Minnesota, 1997.