그리드 컴퓨팅 환경에서의 D-클래스 계산 병렬 알고리즘

신철규, 한재일
국민대학교 전산과학과
e-mail: supply8@cs.kookmin.ac.kr

D-Class Computing Parallel Algorithm the on Grid Computing Environment

Chul-Gyu Shin, Jae-Il Han
School of Computer Science, Kookmin University
e-mail: supply8@cs.kookmin.ac.kr

요 약

D-클래스의 계산은 NP-완전 문제로서 그 결과를 개인키, 공개키로 이용하여 보안에 응용될 수 있는 가능성을 가지고 있으나 계산 복잡도로 인해 현재 극히 제한된 크기의 행렬에 대한 D-클래스만이 알려져 있다. 이 문제를 해결하기 위해 D-클래스 계산을 효율적으로 할 수 있는 수식과 알고리즘을 설계 및 구현하였지만, 행렬의 크기가 증가함에 따라 결과를 얻는 것에는 한계가 있다. 이것을 해결하기 위해 많은 컴퓨터를 사용할 수 있는 그리드 컴퓨팅이 필요하다.

본 논문은 그리드 컴퓨팅 환경에서 최적화된 알고리즘 설계 및 구현을 위해 Globus 가 설치된 클러스터를 구축하고, MPICH 를 이용 효율적인 D-클래스의 계산 알고리즘을 설계 및 구현하여 실행 결과에 대해 논한다.

1. 서론

원소가 0 과 1 값을 가지는 모든 n × n 불리언 행렬에서 특정 관계(relation) {반사성(reflexive), 대칭성(symmetric), 전이성(transitive)} 에 따라 동치(equivalent) 관계에 있는 n × n 행렬의 집합을 D-클래스로 정의한다. D-클래스의 계산은 NP-완전 문제로서 개인키, 공개키로 이용하여 보안에 응용될 수 있는 가능성을 가지고 있으나 계산 복잡도로 인해 현재 극히 제한된 크기의 행렬에 대한 D-클래스만이 알려져 있다[1,2,3].

계산 복잡도가 $O(2^m)$이 되어 NP-완전 문제인 D-클래스의 연산은 그 동안 여러 가지 테스트를 통해 행렬의 크기가 커짐에 따라 생기는 문제들을 해결하고, 연산 속도를 향상시켰다. 하지만, 알고리즘과 수식의 향상만으로 5 × 5 이상의 행렬에 대해 결과를 얻기에는 한계가 있다. 이런 이유로 가상 슈퍼컴퓨터라고 할 수 있는 그리드를 이용할 필요성이 있다.

그리드는 기존의 웹과 차세대 인터넷의 징검다리 역할을 한다는 뜻으로 이런 이름을 붙였으며, 적게는 수십 대에서 많게는 수백만 대의 퍼스널컴퓨터(PC)를 하나의 네트워크로 묶어 제어할 수 있어 가상 슈퍼컴퓨터로 불리는 것이다. 현재 국내에서는 6 개의 슈퍼컴퓨터와 5 개의 클러스터가 그리드 컴퓨팅을 위해서 이용 가능하다.

위의 시스템은 MPICH-G2 를 제공한다. 이것은 Globus toolkit 이 제공하는 service 를 사용할 수 있도록 모든 MPI v1.1 표준 및 MPI v2.0 표준의 일부를 구현한 것으로 그리드 컴퓨팅을 위해 사용될 수 있다.

위의 시스템을 이용한 최적화된 알고리즘 구현 및 설계하여 그리드 컴퓨팅을 하기 위해 먼저 그리드 사용을 위한 Globus 설치후 41 대의 컴퓨터를 이용하여 클러스터를 구축하고 MPICH 를 이용하여 클러스터 상에서 최적화된 알고리즘을 설계 구현하였다[1,2,3,4].

본 논문은 제 2 장에서 그 동안 설계 구현한 순차, 병렬 D-클래스 계산 알고리즘에 대해 살펴보고, 3 장에서는 클러스터를 이용한 알고리즘 설계와 실행 결과에 대해 논하며, 4 장에서는 결론 및 향후 과제를 기

술한다.

```
Initialize D_A , S_B  to an empty set

for each A in M_n(F)
   for each U in M_n(F)
      T = UA
      for each X in M_n(F)
         if TX in M_n(F)
            insert TX to S_B

   for each B in M_n(F)
      if B is in S_B
      for each V in M_n(F)
         T_1 = VB
         for each Y in M_n(F)
            if A equal T_1 Y
               insert B to D_A
               remove B from M_n(F)
```

[그림 1] 알고리즘 1

2. 순차, 병렬 D-클래스 계산 알고리즘

[정의 1]

$$D_A = \{ B \in M_n(F) : \exists C, X, Y, U, V \in M_n(F) \text{ such that } UAX = B, VBY = A \}$$

[정의 1]은 D-클래스를 정의한 것이다. [정의 1]을 기반으로 D-클래스를 계산하는 순차 적으로 연산하는 알고리즘이 [그림 1]이며, $O(2^{4m})$의 연산을 실행해야 된다.

[그림 1]의 알고리즘을 기반으로 메모리를 공유하는 두 프로세서와 메모리를 공유하지 않는 두 프로세서를 이용한 D-클래스 계산 알고리즘을 설계 및 구현하였었다. 메모리를 공유하며 하이퍼 쓰레딩이 되는 두 프로세서 중 3/4 을 이용한 알고리즘으로 UAX 연산을 각 프로세서에서 1/3 씩 처리하여 결과값을 저장 후 VBY 연산을 각 프로세서에서 1/3 씩 처리하였다. 이 알고리즘의 경우 작업분배는 잘 되지만, 행렬의 크기가 커질수록 공유 되는 메모리가 커져 문제점이 발생된다.

이것을 해결하기 위해 두 프로세서로 구성되는 분산 메모리를 이용한 병렬 알고리즘을 설계 및 구현했었다. 첫 번째는 소켓을 이용한 병렬처리 알고리즘으로, 하나의 프로세서는 UAX 연산을 다른 하나는 VBY 연산을 하고, 각 결과 값은 소켓통신을 하여 전달 되어 진다. 두 번째는 SAN 기반의 데이터 공유 기술을 이용한 병렬처리 알고리즘으로 첫 번째와 같이 하나의 프로세서는 UAX, 다른 하나는 VBY 연산을 처리하게 되고, 결과값은 SAN 기반의 데이터 공유 기술을 이용하여 전달된다.

위의 과정을 통해 연산속도 및 문제점들이 향상 되었으나, 5 × 5 이상의 행렬의 대해서 결과를 얻기에는 한계가 있다[1,2,3,4].

3. 클러스터 컴퓨팅 환경에서 MPI 를 이용한 알고리즘

```
Initialize D_A , S_B  to an empty set

rank process num
Start = (my process num-1) * M_n(F)/ 40
End = my process num * M_n(F) / 40 -1

for each A in M_n(F)
  if(process num is 0)
    receive data from 40 process and write to S_B
    send S_B data to process of my process + 1
  else if(process num is 1-40)
    fork()
      loop(if n is from Start to End)
        for each A in M_n(F)
          for each U in M_n(F)
          T = UA
            for each X in M_n(F)
              if TX in M_n(F)
                insert TX to S_B


  if( process num is 1)
    send S_B data to process of my process + 1
    receive S_B data from  process of my process - 1 and
      write to S_B
  else if( process num is 2-39)
    receive S_B data from  process of my process - 1 and
      write to temp
    merge S_B , temp
    send S_B data to process of my process + 1
    receive S_B data from  process of my process - 1 and
      write to S_B
  else if(process num is 40)
    receive S_B data from  process of my process - 1 and
      write to temp
    merge S_B , temp
    send S_B data to process of my 0 process
    receive S_B data from  process of my process - 1 and
      write to S_B


  for each B in M_n(F)
    if(process num is 0)
      receive S_B data from 40 process and write to SB

    if B is in S_B
      fork()
        if(process num is 2-39)
          receive check message from process of my
          process – 1
          if(check message is stop message)
            loop stop
            insert B to D_A
            remove B from M_n(F)
          if(Sendflag is not 1)
            send stop message to process of my process+1
```

위에서 언급했듯이 행렬의 크기가 증가함에 따라 결

```
else if( process num is 1)
   receive check message from 40 process
   if(check message is stop message)
      loop stop
      insert B to D_A
      remove B from M_n(F)
   if(Sendflag is not 1)
      send stop message to process of my process+1
else if( process num is 40)
   receive check message from 19 process
   if(check message is stop message)
      loop stop
      insert B to D_A
      remove B from M_n(F)
   if(Sendflag is not 1)
      send stop message to 1 process

loop(if n is from Start to End)
   for each V in M_n(F)
      T1 = VB
      for each Y in M_n(F)
         if A equal T_1Y
            insert B to D_A
            remove B from M_n(F)
            send stop message to process of my process + 1
            Sendflag is 1

if( process num is 1)
   send S_B data to process of my process + 1
   receive data from 0 process and write to S_B
else if( process num is 2-39)
   receive data from  process of my process - 1 and write to temp
   merge S_B , temp
   send S_B data to process of my process + 1
else if(process num is 40)
   receive data from  process of my process - 1 and write to temp
   merge S_B , temp
   send S_B data to process of my 0 process
   receive data from 0 process and write to S_B
```

[그림 2] 알고리즘 2

과를 얻는 것에는 한계가 있다. 이것을 해결하기 위해서는 계산량을 줄이고, 연산을 더 빠르게 할 수 있도록 해야 된다. 이 중 연산을 빠르게 할 수 있는 해결방안으로 많은 프로세서를 이용하면 될 것이며, 이것은 그리드 컴퓨팅을 하여 해결할 수 있다. 그리드 컴퓨팅환경에서의 최적화된 알고리즘 설계 및 구현하기 위해 먼저 클러스터에서 [그림 1]의 알고리즘을 기본으로 MPI 를 사용한 [그림 2]의 알고리즘을 설계 및 구현 하였다[5].

클러스터는 컴퓨터의 Processor 가 Intel Pentium4 2.66GHz, L1 Cache: 20KB, L2 Cache: 512KB 로 된 것이며, 총 41 대로 구성되었다. 그 중 1 대의 컴퓨터에 Globus 와 MPI 설치되었으며, 이 컴퓨터에서는 데이터를 받아서 처리해주게 된다. 나머지 40 대의 컴퓨터는 각각 1/40 로 나눠진 연산작업을 처리하게 되며, 처리된 데이터를 Globus 가 설치 된 컴퓨터에 전송해준다.

[그림 2]의 프로그램이 실행되어 지면, 각 프로세서에서는 동시에 같은 코드를 실행 하게 된다. 먼저 각 프로세스의 ID 가 할당이 되고, 각 프로세서의 ID 는 0 부터 시작하여 1 씩 증가하며 할당된다. 할당된 프로세서 ID 를 이용하여 각 프로세서에 연산을 배분해주게 된다. 작업 배분 방식은 전체 루프의 index 를 40 으로 나누고, ID 가 1 번인 프로세서에 루프 index 가 0 ~ 1/40 의 작업, ID 가 2 번인 프로세서는 index 가 1/40 ~ 2/40 번의 방식으로 40 번 프로세서까지 나눠주게 된다. 이렇게 배분된 작업을 각 프로세서에서 UAX 연산 후 그 결과를 이용하여 VBY 연산하게 된다.

먼저 fork()를 하여 Child 프로세스 생성 UAX 연산을 하며, Parent 프로세스에서는 Child 프로세스의 결과와 자신의 프로세서보다 ID 가 하나 적은 프로세서에서의 결과를 받아서 두 결과를 비교 자신의 프로세서보다 ID 가 하나 큰 프로세서로 값을 전달 해주는 LINE 을 사용하게 된다. 이렇게 ID 가 40 번인 프로세서까지 값이 전달되며 ID 가 40 번인 프로세서까지 전달되면서 최종적인 결과값을 얻는다. 최종 결과값은 ID 가 0 번 프로세서에게 전달 되며, LINE 방식을 이용 ID 가 0 번부터 40 번인 프로세서까지 전달된다.

전달 받은 값을 사용하여 VBY 연산을 하게 되며, VBY 연산도 UAX 연산 부분처럼 fork()를 하여 Child 프로세스를 생성하고 VBY 연산을 하게 된다. Parent 프로세스에서는 VBY 의 연산 결과를 기다리며, A 와 B 가 D-클래스가 된다는 결과를 얻게 되면, LINE 방식을 이용하여 모든 프로세서에게 현재 작업을 중단하며, 다음 작업을 실행할 수 있게 메시지를 보내고 받는 것을 수행한다. 이렇게 모든 작업을 마치게 되면 그 결과는 다시 LINE 방식을 이용하여 결과를 전달 병합하여 최종적인 결과는 ID 가 0 번인 프로세서로 전달 된다.

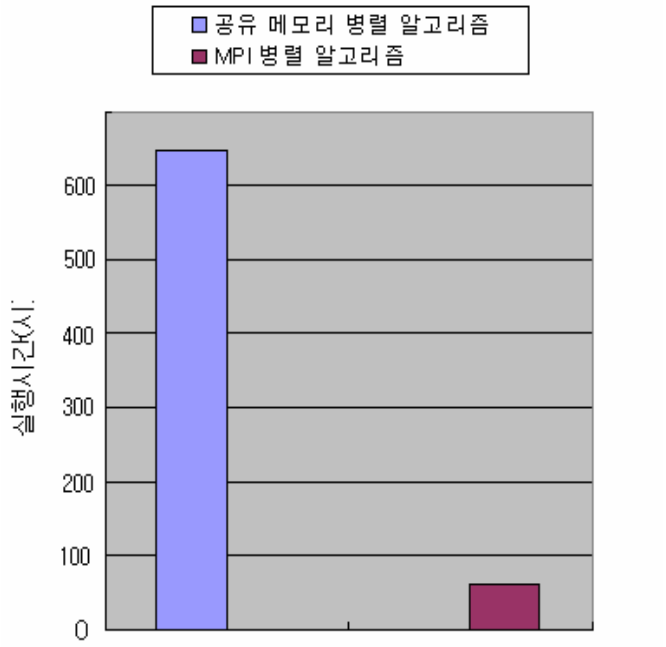
[그림 3]은 4 x 4 행렬에서 그 동안의 실행 결과 중 가장 결과가 좋았던 공유 메모리 병렬 알고리즘과 MPI 병렬 알고리즘의 실행 시간을 보여주고 있다. 공유 메모리 병렬 알고리즘의 실행 컴퓨터의 프로세서는 Intel Xeon 2.66(FSB533), L1 Cache: Execution Trace cache, L2 Cache: 512KB Advanced Transfer 로 구성 된 것이다. 전체 성능 중 3/4 을 사용하여 코어 속도는 전체적으로 약 4GHz 정도라고 할 수 있다. 그리고 MPI 병렬 알고리즘을 실행한 클러스터의 총 코어 속도는 약 106GHz 이다. 다른 부분도 연산에 적지 않은 영향을 미치지만, 본 논문의 연산에 가장 많은 영향이 미치는 코어 속도이다. 코어 속도의 차이에 따른 실행 시간을 비교하면 약 26 배 향상되었지만, 실행 시간은 1/11 정도 줄었다. 이런 결과는 작업을 분배하여 연산 후 결과를 받고 그것을 처리 하는 과정이 있다. 이 때 프로세서 사이에 값을 전달 하는 동안 프로세서가 연산을 하지 않는 문제점이 발생한다.

4. 결론 및 향후 과제

D-클래스의 계산은 NP-완전 문제로서 그 결과를 개인키, 공개키로 이용하여 보안에 응용될 수 있는 가능성을 가지고 있다. 하지만 계산 복잡도가 NP-완전 문제이기 때문에 현재 극히 제한된 크기의 행렬에 대한 D-클래스만이 알려져 있다. 이 문제를 해결하기 위해 D-클래스 계산을 효율적으로 할 수 있는 수식과 알고리즘을 설계 및 구현하였지만, 행렬의 크기가 증가함에 따라 결과를 얻는 것에는 한계가 있다. 이런 문제점을 해결하기 위해 많은 컴퓨터를 사용할 수 있는 그리드 컴퓨팅이 필요하다. 본 논문에서는 그리드 컴퓨팅 환경에서 MPICH-G2 를 사용하여 최적화된 병렬 알고리즘을 설계 및 구현하기 위해 Globus 와 MPI 가 설치된 클러스터 컴퓨터에서 병렬 알고리즘을 설계 및 구현 하여 실행 결과를 보여주었다.

클러스터 컴퓨터에서 MPI 를 이용하여 연산한 실행 시간은 공유 메모리에서의 병렬 알고리즘과 비교해 많은 향상이 있었다. 하지만 프로세서의 수가 늘어나 전체적으로 시스템이 향상된 결과만큼 실행 시간이 단축 되지는 않았다. 이런 문제를 해결하기 위해 각 프로세서 사이에서 전송되어야 할 데이터와 전송 시간을 줄일 수 있도록 해야 될 것이다.

본 논문에서 야기되었던 문제를 해결하고 그리드 컴퓨팅을 하기 위하여, 현재 모든 행렬을 연산하는 것이 아니라 행렬의 특성에 따라 부분집합으로 나누고, 나눠진 부분집합을 연산하여 D-클래스를 얻는다. 그렇게 얻어진 부분집합의 D-클래스들을 서로 비교하는 방법으로 그리드 컴퓨팅 환경에서 MPICH-G2 를 이용하여 실행할 수 있는 병렬 알고리즘을 설계 및 구현하고 있다[6,7,8,9,10,11,12].



[그림 3]

참고문헌

[1] Chul-Gyu Shin, Jae-Il Han, “A Parallel Algorithm the for D-Class Compution “, KIPS Fall Conference, 2004

[2] Chul-Gyu Shin, Jae-Il Han, “A Study on the D-Class Computing Algorithm “, KIPS Spring Conference, 2004, pp. 903-906

[3] Dock Sang Rim, Jin Bai Kim, “Tables of D-Classes in the semigroup Bn of the binary relations on a set X wit n-elements,” Bull. Korea Math. Soc. Vol.20. No. 1, 1983, pp. 9-13

[4] Kyle Loudon, Algorithms with C, O’Reilly, 2000

[5] Barry Wilkinson, Michael Allen, Parallel Programming with MPI, Prentice Hall, 1999

[6] Gene H. Golub, Charles F. Van Loan. Matrix Computation, The Johns Hopkins’ University Press, 1983

[7] Kim K. Butler, “On (0, 1)- matrix semigroups,” Semigroup Forum 3, 1971, pp. 74-79

[8] Ki Hang Kim, F. Roush, “Generalized fuzzy matrices,” Fuzzy sets and systems 4, 1980

[9] John Gunnels 외 3 인, “A Flexible Class of Parallel Matrix Multiplication Algorithms, ” Department of Computer Sciences The University of Texas at Austin

[10] F. Thomson Leighton, Parallel Algorithms And Architectures: Arrays•Trees•Hypercubes, Morgan Kaufmann, 1992

[11] Fox, G.., S. Otto, and A. Hey, “Matrix algorithms on a hypercube I: matrix Multiplication,” Parallel Computing 3, 1987, pp. 17-31.

[12] Fran Berman, Geoffrey C. Fox, Anthony J. G. Hey, “Grid Computing-Making the Global Infrastructure a Reality, WILEY, 2001