내부 순환문 개선을 통한 Linux 기반의 D-클래스 계산 고효율 순차 알고리즘

신철규, 한재일
국민대학교 전산과학과
e-mail: M2004049@cs.kookmin.ac.kr

Serial Algorithm for D-Class computation with an Improved Inner Loop on the Linux Platform

Chul-Gyu Shin, Jae-Il Han
School of Computer Science, Kookmin University
e-mail: M2004049@cs.kookmin.ac.kr

요 약

D-클래스는 보안에 응용될 수 있는 가능성을 가지고 있으나 D-클래스의 계산은 NP-완전문제로서 행렬크기 증가에 의한 연산 량 증가 문제 해결을 위해 병렬 컴퓨팅 환경에서의 병렬 알고리즘 설계 및 구현이 필요하다. 본 논문은 그리드 컴퓨팅 환경에서의 D-클래스 계산을 위해 Globus 가 설치된 클러스터를 구축하고, MPICH를 이용 효율적인 D-클래스 계산 알고리즘을 설계 및 구현, 실행 결과 그리고 연산 량을 줄일 수 있는 수식 연구와 연구한 수식에 기반한 순차 알고리즘을 논한다.

1. 서론

그리드는 동시에 여러 사이트와 연결해 정보를 주고받을 수 있는 수평 구조로 되어 있다. 즉 지구 반대편에 있는 사람의 컴퓨터와 연결해 동시에 같은 연구를 수행할 수도 있고, 한 사람이 찾은 자료를 여러 사람이 동시에 보면서 서로 의견을 나눌 수도 있다. 나아가 한 대의 컴퓨터가 여러 곳에 흩어져 있는 컴퓨터를 원격 조정해 복잡한 계산을 나누어 시킨 뒤 이들을 다시 하나로 합칠 수도 있다.

이런 그리드 시스템을 이용하여 그리드 컴퓨팅을 할 수 있으며, 그리드 컴퓨팅은 다음의 이유 때문에 일반적인 추세가 된다[1].

- 주어진 량의 컴퓨터 자원을 비용편익 측면에서 가장 효율적으로 만들 수 있는 능력
- 엄청난 량의 컴퓨팅 능력 없이는 풀기 어려운 문제들을 해결할 수 있는 하나의 방편
- 많은 수의 컴퓨터들을 하나의 공동목표를 위한 상승적인 협동작업으로 이용하고 관리할 수 있음

그리고, 현재 국내에서는 6 개의 슈퍼컴퓨터와 5 개의 클러스터가 그리드 컴퓨팅을 위해서 이용 가

능하다.

원소가 0 과 1 값을 가지는 모든 n × n 불리언 행렬에서 원소들의 관계가 특정 관계(relation) 에 따라 동치(equivalent) 관계에 있는 n × n 행렬의 부분 집합을 D-클래스로 정의한다[2]. D-클래스는 보안에 응용될 수 있는 가능성을 가지고 있으나 D-클래스의 계산은 NP-완전문제로서 행렬크기 증가에 의한 연산 량 증가로 현재 극히 제한된 크기의 행렬에 대한 D-클래스만이 알려져 있다[2,3,4].

D-클래스의 연산은 계산 복잡도가 $O(2^m)$이 되어 NP-완전 문제이다. 이런 이유로 가상 슈퍼컴퓨터라고 할 수 있는 그리드를 이용할 필요성이 있고, 그리드 컴퓨팅을 위해 Globus 를 설치하고 41 대의 컴퓨터를 이용하여 클러스터를 구축하여 병렬 알고리즘을 설계 및 구현 하였다. 또한 연산량을 줄여 고효율의 순차 알고리즘을 설계 및 구현 하였다. 고효율 순차 알고리즘은 그 연구된 병렬 알고리즘보다 향상된 결과를 보인다. 따라서 이 알고리즘을 그리드 컴퓨팅 할 필요가 있고, 그동안 구현된 알고리즘의 문제점을 분석하여, 고효율의 병렬 알고리즘을 설계 및 구현 할 수 있도록 참고 한다.

병렬 알고리즘의 문제점을 분석하기 위해, 먼저 알고리즘을 살펴보고 원인과 결과를 분석한다 [2,3,4,5].

본 논문은 제 2 장에서는 연구된 병렬 D-클래스 계산 알고리즘에 대해 살펴보고 문제점을 분석하며, 3 장에서는 수식 연구를 통한 연산 량을 줄인 알고리즘 설계와 실행 결과에 대해 논하며, 4 장에서는 결론 및 향후 과제를 기술한다.

2. 관련 연구

2.1 순차 알고리즘

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

[정의 1]

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

같이 단일 프로세서 만으로 NP-완전 문제인 D-클래스의 연산을 하는 것은 비 효율적이다.

2.2 두 프로세서를 이용한 병렬 알고리즘

효율성 문제를 개선 하기 위하여 [그림 1]의 알고리즘을 기반으로 메모리를 공유하는 두 프로세서와 메모리를 공유하지 않는 두 프로세서를 이용한 D-클래스 계산 알고리즘을 설계 및 구현 하였다. 메모리를 공유하며 하이퍼쓰레딩[5]이 되는 두 프로세서를 이용한 알고리즘으로 각 프로세서에 UAX 연산의 1/3 을 분배하고, 처리하여 결과값을 저장한다. 저장된 결과 B 를 이용하여 VBY 연산하고, 각 프로세서에서 1/3 씩 처리하였다. 이 알

```
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_1Y
               insert B to D_A
               remove B from M_n(F)
```

[그림 1] 알고리즘1

고리즘의 경우 작업분배는 잘 되지만, 행렬의 크기가 커질수록 공유 되는 메모리가 커져 문제점이 발생된다. 즉, 행렬의 크기가 커질수록 계산 중간에 공유되는 정보가 기하급수적으로 증가하여 공유 메모리 MIMD 구조에서 문제점이 야기 된다. 따라서, 분산 메모리 MIMD 구조에서의 병렬 알고리즘이 필요하다

이것을 해결하기 위해 두 프로세서로 구성되는 분산 메모리 구조에서 병렬 알고리즘을 설계 및 구현 했다. 첫 번째는 소켓을 이용한 병렬처리알고리즘으로, 하나의 프로세서는 UAX 연산을 다른 하나는 VBY 연산을 하고, 각 결과 값은 소켓통신을 하여 전달 된다. 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 과정에서 연산속도가 향상 되었으나, 행렬 크기의 증가에 따른 연산 량 증가에 의해 5 × 5 이상의 행렬에서 D-클래스의 결과를 얻기에는 한계가 있다[2,3,4,5].

2.3 그리드 클러스터 기반 병렬 알고리즘

행렬의 크기 증가로 연산 량이 증가하여 D-클래스 연산 결과를 얻는 것에는 한계가 있다. 이것을 해결하기 위해 연산 속도를 향상 시키는 방법과 연산 량을 줄일 수 있도록 고려한다. 이 중 연산 속도를 향상 시키는 방안은 많은 프로세서를 이용하여 병렬처리 하며, 프로세스 수의 증가로 속도를 향상 시킬 수 있다. 많은 자원을 이용하는 것은 그리드를 이용하여 해결 할 수 있다. 그리드 컴퓨팅환경에서의 병렬 알고리즘 설계 및 구현하기 위해 Globus 가 설치 된 클러스터 환경에서 [그림 1]의 알고리즘을 기반으로 MPICH 를 사용한 알고리즘을 설계 및 구현 하였다[6]. 클러스터를 구성하는 컴퓨터의 Processor 는 Intel Pentium4 2.66GHz(L1 Cache: 20KB, L2 Cache: 512KB) 이며, 총 41 대로 구성되었다. Globus 가 설치된 컴퓨터에서 프로그램이 시작되며, 각 프로세서들에서 처리된 결과 받고, 결과를 처리하여 각 프로세서들에게 전송하는 역할을 한다.

다른 40 대의 컴퓨터는 연산을 하며, 각 프로세서에서 전체 연산의 1/40 의 연산을 처리하게 된다. 각 프로세서에서 처리된 연산 결과는 Globus 가 설치 된 컴퓨터에 전송한다. 프로그램이 실행되면 각 프로세서에 ID 가 할당 되며, ID 는 0 부터 시작, 1 씩 증가하며 할당된다. ID 를 기준으로 각 프로세서에 연산이 배분되어 할당 된다. 연산의 배분은 각 프로세서당 전체 연산 량의 1/40 이 된다.

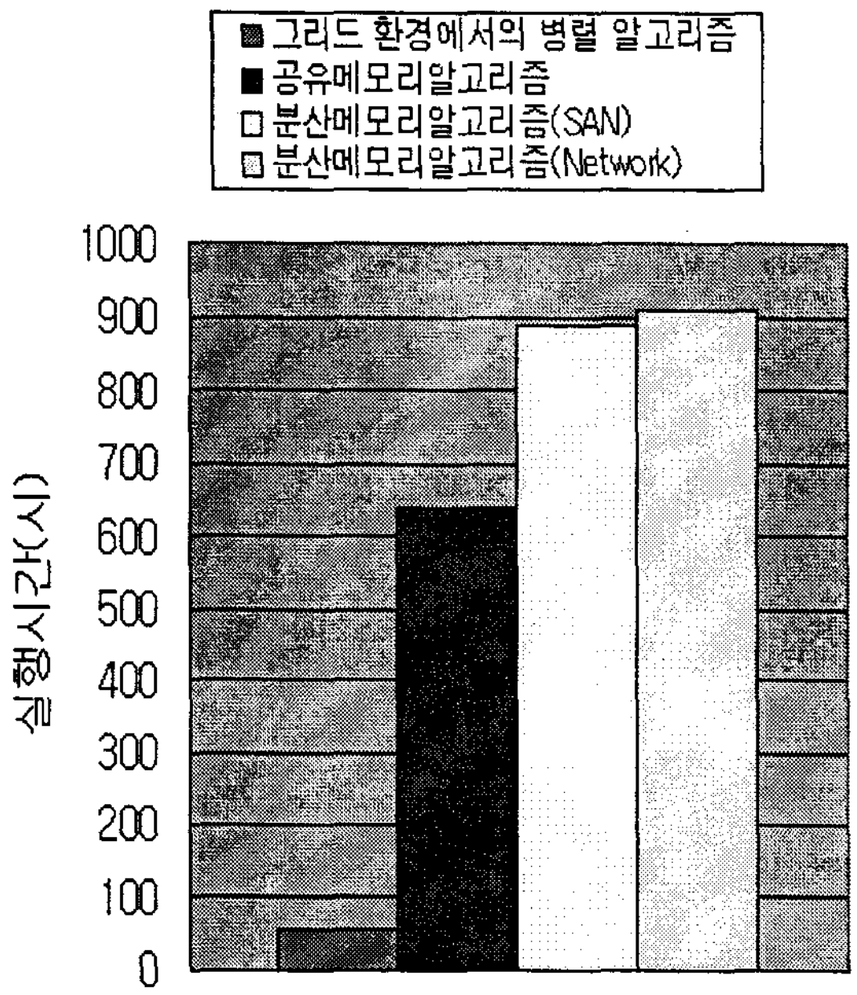
각 프로세서에 배분된 연산에 따라 연산을 하게 되며, UAX 연산 과정은 fork()를 하여 Child 프로세스를 생성 하여 Child 프로세서에서 UAX 연산을 한다. 그리고, Parent 프로세스에서는 Child 프로세스의 결과와 자신의 프로세서보다 ID 가 1 작은 프로세서에서의 결과를 받아서 두 결과를 비교, 자신의 프로세서보다 ID 가 1 큰 프로세서로 값을 전달 하는 LINE 을 사용하게 된다. 각 결과는 ID 가 1 번인 프로세서부터 ID 가 40 번인 프로세서까지 전달 된다. 최종 결과값은 ID 가 0 번인 프로세서로 전달 되며, 다음 과정인 VBY 연산을 위해 필요한 UAX 의 연산 결과를 LINE 방식을 이용 ID 가 0 번부터 40 번인 프로세서까지 전달된다.

다음 과정인 VBY 연산도 UAX 연산 부분처럼 fork()를 하여 UAX 에서의 연산과정과 동일하게 Child 프로세스와 Parent 프로세서로 분리하여 처리된다.

2.4 병렬 알고리즘 실행 결과

[그림 2]는 4 × 4 행렬에서 병렬 알고리즘의 실행 결과를 보여준다. 공유 메모리 병렬 알고리즘은 프로세서가 Intel Xeon 2.66(FSB533)(L1 Cache: Execution Trace cache, L2 Cache: 512KB Advanced Transfer)으로 2 개의 프로세서로 구성되어 있다.

전체 성능 중 3/4 을 사용하여 코어 속도는 전체적으로 약 4GHz 정도라고 할 수 있다. 분산 메모리 병렬 알고리즘은 프로세서가 Processor 는 Intel Pentium4 2.0GHz(L1 Cache: 8KB, L2 Cache: 128KB)로 두 대의 컴퓨터를 사용하여. 코어 속도는 4GHz 이다. 그리고 그리드 환경에서의 병렬 알고리즘을 실행한 클러스터의 총 코어 속도는 약 106GHz 이다. 4 × 4 이하의 D-클래스에 대한 [그림 1] 알고리즘은 CPU-Bound 프로그램으로 cache size, Internal bus, External bus 의 frequency 등 다른 부분도 연산에 적지 않은 영향을 미치지만[8], 가장 많은 영향이 미치는 코어 속도이다. 두 알고리즘이 테스트된 코어 속도를 비교 하면, 그리드 환경에서의 병렬 알고리즘은 공유, 분산 메모리 병렬 알고리즘 보다 약 26 배 향상되었다. 그에 반해 실행 시간은 공유 메모리 병렬 알고리즘 약 1/11, 분산 메모리 알고리즘 약 1/16 정도 향상 되었다. 시스템 성능의 향상만큼 효율성이 향상되지 않았으며, 원인은 프로세서의 수의 증가하여 연산 결과를 모든 프로세서에서 공유하는데 소비 되는 시간이 증가하고 데이터 공유 과정에서는 연산 작업이 실행 되지 않기 때문이다. 이런 문제점은 고효율 순차 알고리즘을 그리드 환경에서 병렬 처리할 때 고려 해야 된다.



[그림 2]

3. 수식 변환을 통한 알고리즘 최적화

문제를 해결 할 다른 방법으로 수식 연구를 하여 연산 량을 줄이는 것이다. 모든 n × n 불리언 행렬의 원소는 2^m (m = n × n) 이 되고, 모든 원소 사이의 연산은 2^m × 2^m 번이 된다. 하지만 이 연산의 모든 결과값은 불리언 행렬의 특성에 따라 불리언 행렬의 연산 결과가 중복되는 것을 고려하여 최적화한 알고리즘을 설계 및 구현 하였다.

Initialize D_A, S_A, S_B to an empty set

for each A in $M_n(F)$
  for each V in $M_n(F)$
    for each Y in $M_n(F)$
      if A equal VY
        insert V to S_A

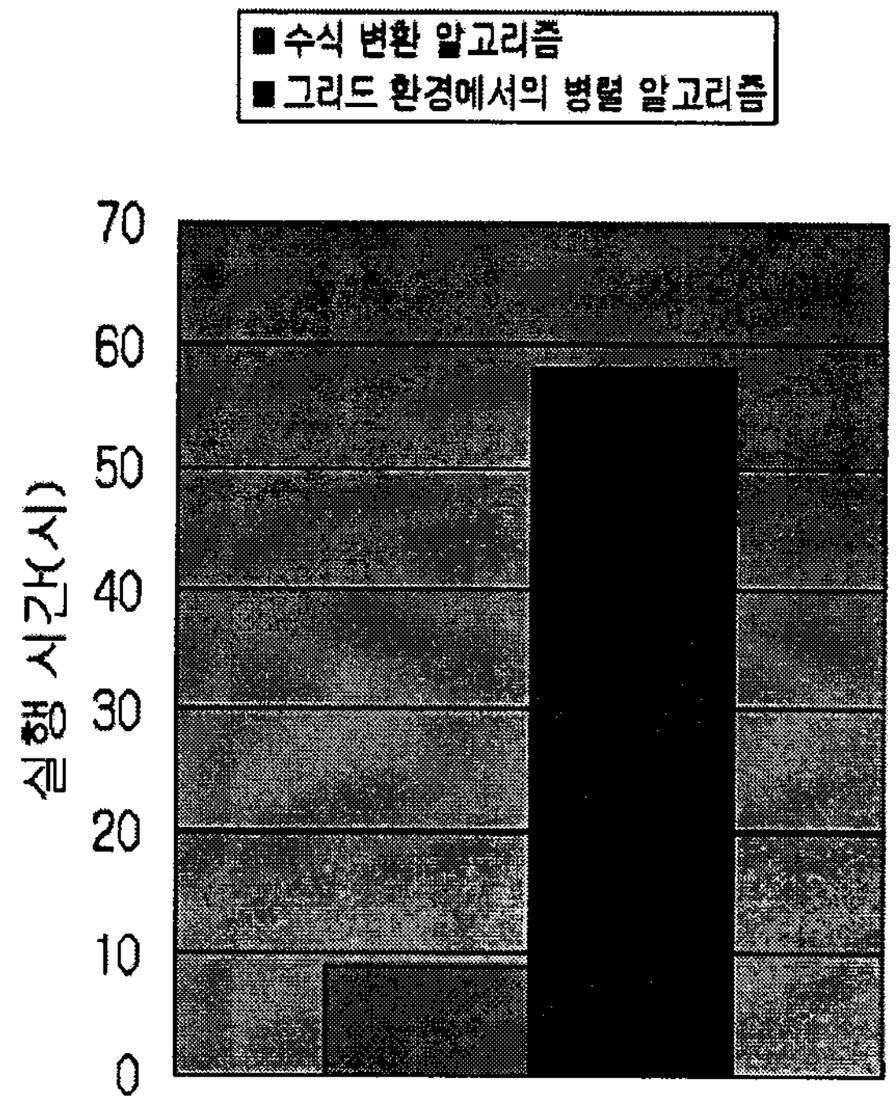
for each U in $M_n(F)$
  $T = UA$
  for each X in $M_n(F)$
    if TX in $M_n(F)$
      insert TX to S_B

for each B in $M_n(F)$
  if B is in S_B
    for each V in $M_n(F)$
      $T_1 = VB$
        if T_1 in S_A
          insert B to D_A
          remove B from $M_n(F)$

[그림 3] 알고리즘3

[그림 1] 알고리즘의 연산은 임의의 A, B 행렬에 대한 D-클래스 연산을 위해, UAX 연산 후 VBY 연산을 하게 된다. 이 알고리즘도 UAX 연산을 통하여 결과에 대한 중복을 제거하며, 다음 과정인 VBY 연산을 하게 된다. [그림 3] 알고리즘의 UAX 연산은 [그림 1] 알고리즘의 UAX 연산방법과 동일하다. VBY 연산과정은 언급된 불리언 행렬의 특징을 적용한다. 먼저 VY 연산 결과를 저장하고, 저장된 데이터를 참고하여 VBY 연산 량을 줄인다.

VBY 연산은 연산 결과가 A 행렬인지를 판단하기 위한 것으로 시간 복잡도는 $O(2^{3m})$이 된다. 하지만, 불리언 행렬의 연산 결과의 특성을 고려하면, VB 연산 결과는 V 에 포함하는 것을 알 수 있다. 따라서, VY 연산 결과가 A 행렬이 되는 V 원소를 S_A 저장하고, UAX 연산 결과로 얻어진 B 와 V 의 연산 결과가 S_A 저장되어 있는지 검사하여 최종 결과를 얻을 수 있다. VBY 연산은 $O(2^{2m})$이 되어 연산 량이 감소되고, 전체 연산은 $O(2^m (2^{2m} + 2^{2m}))$이 되어, $O(2^{3m})$이 된다. Intel Xeon 2.66 (FSB533) (L1 Cache: Execution Trace cache, L2 Cache: 512KB Advanced Transfer)의 프로세서로 연산을 실행 했으며, 그리드 환경에서의 병렬 알고리즘과 실행 시간을 비교 하였다. [그림 4]에서 결과를 보여 주고 있으며, 41 개의 프로세서를 이용한 그리드 환경에서의 병렬 알고리즘 보다 프로세서 하나를 이용한 순차알고리즘이, 약 9.5 배 성능이 향상되었다. 이런 결과는 연산 량이, $O(2^{4m})$과 $O(2^{3m})$으로 차이가 나기 때문이다.



[그림 4]

4. 결론

D-클래스는 보안에 응용될 수 있는 가능성을 가지고 있으나 D-클래스의 계산은 NP-완전문제로서 행렬크기 증가에 의한 연산 량 증가로 현재 극히 제한된 크기의 행렬에 대한 D-클래스만이 알려져 있다

이런 D-클래스 계산을 효율적으로 할 수 있도록 수식의 연구와 병렬 알고리즘을 설계 및 구현하였다. 하지만, 행렬의 크기 증가에 따라 연산 량의 증가로 D-클래스를 연산에 한계가 있다. 이런 문제를 해결하기 위해 그리드 컴퓨팅과 연산 량을 줄인 알고리즘이 필요하다. 본 논문에서는 구현된 병렬 알고리즘과 Globus 와 MPICH 가 설치된 클러스터 환경에서 병렬 알고리즘, 그리고 수식 연구를 통한 연산 량을 줄인 알고리즘을 설계 및 구현하고 실행 결과를 보여주었다.

그리드 환경에서의 병렬 알고리즘의 실행 시간은 공유 메모리에서의 병렬 알고리즘과 비교해 향상 되었다. 하지만 시스템의 향상만큼 실행결과는 향상되지 않았다. 이런 문제를 해결하기 위해 각 프로세서간 전송되어야 할 데이터와 전송 시간을 줄여야 한다. 수식 연구로 연산 량을 $O(2^{4m})$에서 $O(2^{3m})$으로 감소시켜 구현된 [그림 3]의 알고리즘도 병렬 처리가 필요하다. 그리드 컴퓨팅환경에서 병렬 알고리즘의 설계 및 구현할 때 위의 문제를

고려해야 된다[9,10,11,12,13,14].

[참고문헌]

[1] grid computing, “http://www.terms.co.kr/Gridcomputing.htm”

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

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

[4] 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

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

[6] Technology, "hyperthreading" ,” "http://www.intel.com/techtrends/technologies/hyperthreading.htm"

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

[8] Jonn L.Hennessy & David A.Patterson, “Computer Architecture (3rd International Edition) : A Quantitative Approach ”, MORGAN KAUFMANN, 2002

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

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

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

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

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

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