MPMD 방식의 비동기 연산을 이용한 응용 수준의 무정지 선형 시스템의 해법

박 필 성†

요 약

대규모 병렬 연산에 있어서, 계산 노드 혹은 이들을 연결한 통신 네트워크의 장애는 연산 실패로 끝나며, 소중한 계산 시간이 낭비된다. 그러나 현재의 MPI 표준은 이에 대한 대안을 제시하지 않고 있다. 본 논문에서는, 비표준의 무정지형 MPI 라이브러리가 아닌 MPI 표준 함수들만을 사용하여, MPMD 방식의 비동기 연산을 도입한 응용 수준의 무정지형 선형 시스템의 해법을 제안한다.

키워드 : 무정지, MPMD, 선형시스템, 비동기 반복, MPI

An Application-Level Fault Tolerant Linear System Solver Using an MPMD Type Asynchronous Iteration

Pil Seong Park†

ABSTRACT

In a large scale parallel computation, some processor or communication link failure results in a waste of huge amount of CPU hours. However, MPI in its current specification gives the user no possibility to handle such a problem. In this paper, we propose an application-level fault tolerant linear system solver by using an MPMD-type asynchronous iteration, purely on the basis of the MPI standard without using any non-standard fault-tolerant MPI library.

Key Words : Fault Tolerant, Multiple Program Multiple Data, Linear System, Asynchronous Iteration, Message Passing Interface

1. 서 론

오늘날 Grand Challenge(http://www.absoluteastronomy/encyclopedia/G/Gr/Grand_Challenge_problem.htm 참조)와 같이 단일 CPU로는 처리가 불가능한 거대한 문제들의 해법이 시도되고 있는데, 이는 대규모 병렬 시스템 혹은 지리적으로 분산된 GRID 환경을 이용하여 수행되고 있다. 그러나 이런 컴퓨팅 환경은 단일 CPU 시스템과는 많은 점에서 다르며 새로운 문제를 많이 제기한다.

이런 대규모 병렬 연산에서 특히 문제가 되는 것은, 일부 노드나 이들을 연결하는 통신 회선의 예기치 못한 장애로 인한 연산실패이다. 이는 계산 시간과 자원의 낭비를 가져오므로, 사용자 입장에서는 응용 프로그램 수준의 무정지(fault tolerance)의 구현이 점차 중요한 문제로 부각되고 있다. 그러나 현재 병렬 연산에 널리 사용되는 MPI(Message Passing Interface)[7] 표준은 이런 문제를 위한 방안을 제시하고 있지 않아, 여러 가지의 무정지형 MPI 라이브러리가 개발되고 있다. 그런데 이들은 아직 표준화가 이루어지지 않아 서로 호환되지 않고 병렬 코드의 이식성이 떨어지며, 이들을 이용할 경우 상당한 시스템 오버헤드가 필요하다[6]. 또한 상당수의 병렬 시스템은 표준 MPI 라이브러리만이 설치되어 있어 무정지형 MPI 라이브러리는 지원되지 않고 있다.

본 논문에서는 어떤 조건을 만족하는 거대 선형 시스템에 적용할 수 있는 무정지형 해법을 제안하고자 한다. 즉 비표준의 무정지형 MPI 라이브러리를 사용하는 대신, 어느 프로세스의 장애를 파악하고 백업 프로세스로 하여금 이를 복구하는 메커니즘을 MPI의 표준 함수만을 사용하여 응용 프로그램의 수준에서 구현함으로써 불필요한 시스템 오버헤드를 줄이고 이식성을 높였다. 또한 비동기 연산을 도입함으로써 복구 과정을 단순화하고 빠른 복구가 가능하도록 하였으며, 다른 프로세스들은 복구 과정과 무관하게 연산을 계속할 수 있도록 함으로써 전체적인 수렴 속도를 높였다.

2. 관련 연구

2.1 SPMD와 MPMD

오늘날 병렬 시스템들은 대부분 MIMD(multiple-instruction-multiple-data) 타입으로 어느 순간에 각 프로세서는 각기

※ 본 논문은 해양수산부 연구개발과제로서 한국해양연구원이 수행중인 해양예보시스템 구축사업의 지원으로 수행되었다.
† 정 회 원 : 수원대학교 IT대학 컴퓨터학과 부교수
논문접수 : 2005년 5월 17일, 심사완료 : 2005년 8월 31일

다른 작업을 수행한다. MIMD형의 연산을 지원하는 프로그래밍 방식은 SPMD(single-program-multiple-data)와 MPMD(multiple-program-multiple-data)로 나눌 수 있다[8].

SPMD 방식은 모든 프로세서들이 동일한 프로그램을 수행하는 것으로, 프로세서별로 분기하여 각기 다른 작업을 한다. 이는 동기화를 자주 시행하며 미리 잘 정의된 시간에 데이터를 교환하는 응용에 적합하며. 일관성을 유지하기 쉽고 프로세서들 간의 상호 작용을 명확히 알 수 있다. 그러나 하나의 프로그램이 모든 프로세서의 역할을 포함하므로 프로그램이 복잡하고 불필요하게 많은 메모리를 필요로 한다.

반면, MPMD 방식은 둘 이상의 프로그램을 작성하여 프로세서별로 적절한 프로그램을 시행토록 하는 것이다. 이는 빠른 개발 사이클, 코드의 높은 재활용률, 모듈 프로그래밍 등의 장점이 있으며, 일정치 않은 계산 부하량이나 불규칙하거나 예측하기 어려운 메시지 전달 패턴을 가진 응용에 적절하다[3]. 또한 클라이언트-서버 타입 환경의 구현에 적절하며, 분기 루틴이 적어 프로그램의 크기가 작고 메모리도 적게 차지한다. 그러나 프로세서들 간의 상호작용에 있어서 일관성 유지가 쉽지 않은 것이 단점이다. MPMD 방식이라고 해서 모든 프로세서들에게 각기 다른 프로그램을 제공하는 것이 아니라 비슷한 작업을 수행하는 프로세서별로 그룹을 만들어 그룹당 하나의 프로그램을 제공하는 것이 일반적이다.

2.2 MPI와 무정지형 MPI 라이브러리들

메시지 패싱(message passing)은 프로세서간에 교환할 데이터를 메시지 전달 함수를 사용하여 주고받는 병렬 연산 모델이다. 메시지 패싱 라이브러리(message passing library)는 이런 함수들의 집합체로서, 이들의 표준을 정한 것이 MPI이다. MPI 환경에서는 계산 주체가 CPU가 아니라 프로세스(process)이며 프로그램 실행시 필요한 프로세스의 개수를 지정하게 된다. 각 프로세스는 랭크(rank)에 의해 구분되며, n개의 프로세스를 사용할 경우 0부터 n-1의 값을 가진다. 어느 프로세스가 송신함수를 호출하면 필히 그에 대응하는 수신함수를 호출하는 프로세스가 있어야 하며, 이 경우 송수신 데이터의 수, 데이터의 형, 태그(tag) 등이 일치해야 하고 상대 프로세스는 랭크를 사용하여 지정한다.

현재의 MPI의 표준은, 무정지형 연산을 지원하지 않으므로 이를 보완하기 위해 개발된 것이 무정지형 MPI 라이브러리이다. 이들은 체크포인트/롤백(checkpoint/roll-back)을 사용하는 것과 메시지나 프로세스의 복제(replication) 기술을 사용하는 두 종류로 나눌 수 있다. 전자의 예로는 Co-Check MPI, MPICH-V, LA-MPI가 있으며, MPI/FT 및 MPI-FT는 후자의 예이다. 그러나 비교적 적은 오버헤드를 가지는 FT-MPI라도 모든 것이 자동으로 복구되어 수행이 계속되는 것이 아니라, 장애 발생시 응용 프로그램으로 하여금 적절히 대처할 수 있도록 하는 도구만 제공할 뿐이며, 이를 응용 프로그램 내에서 구현하여 복구하고 연산이 계속되도록 하는 것은 사용자의 몫이다[6].

2.3 비동기 반복 알고리즘

병렬 연산의 주류는 동기 알고리즘(synchronous algorithm)으로 동기화가 필요하며 부하 균형이 필수적이다. 그러나 문제에 따라 부하 균형이 어렵거나, 사용되는 프로세서들의 성능이 서로 다르거나, 네트워크 전송 지연 등으로 동기화로 인한 유휴시간이 길어질 수밖에 없다.

비동기 반복(asynchronous iteration) 알고리즘은 유휴 시간으로 인한 성능 저하를 완화하는 하나의 방법으로, Chazan & Miranker[4]의 선구적인 연구이래, 선형 시스템의 해법으로 많은 연구가 이루어졌으며[2, 9, 10, 11], 일반적인 전산학 문제로 응용 영역이 확대되고 있다[5, 12]. 이상적인 비동기 알고리즘의 경우, 각 프로세서는 유휴시간이 거의 없이 자신의 작업을 수행하며 다른 프로세서로부터 데이터가 오지 않으면 기다리지 않고 이전의 데이터를 사용하여 연산을 계속한다. 따라서 실제 수행 시간(wall clock time)은 동기 알고리즘보다 1/3-1/2까지 단축된 결과도 보고되고 있다[2, 9].

동기 알고리즘을 사용하면 직렬 연산과 동일한 결과를 얻을 수 있어서 수렴이 보장되나, 비동기 알고리즘은 수행시마다 다른 결과를 얻게 되므로 원하는 해로의 수렴여부는 불투명하다. 그러나 선형 시스템 해법의 경우, 비동기 반복 알고리즘의 수렴이 보장되기 위한 행렬의 조건은 다음과 같이 알려져 있다[11].

Theorem 1. 선형 시스템 $A\bar{x} = \bar{b}$ 에서, A^{-1}와 모든 대각 블록 A_{ii}^{-1}의 모든 성분들이 음이 아니며(nonnegative)이며 비대각 블록(off-diagonal block) $A_{ik}, i \neq k$의 모든 성분들이 양이 아니면(nonpositive) 어떤 벡터를 초기 벡터 $\bar{x}$로 택하여 계산을 시작하여도 항상 정확한 해로 수렴한다.

이 정리를 만족하는 선형 시스템의 해법에 비동기 반복 알고리즘을 적용할 경우, 각 프로세스는 완전히 서로 무관하게 비동기적으로 연산을 수행하여도 항상 옳은 답으로 수렴하게 된다는 것을 의미하는데, 이 정리는, 본 논문에서 핵심적인 역할을 한다. 그런데 이 정리가 요구하는 조건은 특수한 성질이 아니라 상당수의 자연과학 계산에서 볼 수 있는 비교적 일반적인 성질로서 본 논문에서는 이 조건을 만족하는 문제를 다루기로 한다.

3. 응용 수준의 무정지형 해법

본 절에서는 대상 문제를 푸는 일반적인 동기 알고리즘을 소개하고, 새로운 무정지형 해법의 고안을 위해 고려할 사항과 이를 구현할 수 있는 도구에 대해 설명하고자 한다. 그리고 비동기 반복 알고리즘을 결합한 무정지형 MPMD 알고리즘에 대해 기술하도록 한다.

3.1 문제 및 동기 알고리즘의 예

본 논문은 [1]에서 사용한 것과 유사한 선형 시스템 $A\bar{x} = \bar{b}$ 의 해법을 다룬다. 행렬 A는 크기가 900만×900만인 거대 희소행렬로서 Theorem 1의 조건을 만족하며, A의 그래프는 3,000×3,000의 2차원 격자문제와 동일하다,

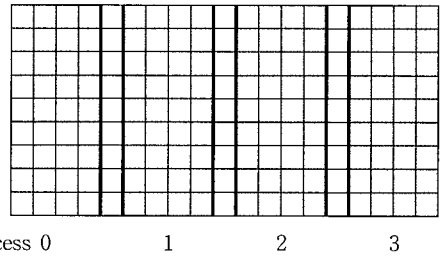
이해를 돕기 위해, 20×10의 격자 문제를 1차원 영역 분할(domain decomposition)을 사용하여 (그림 1)과 같이 4개의

프로세스로 블록 자코비법(block Jacobi)에 의해 연산하는 경우를 보자. 즉 각 프로세스는 수직격자선 5개를 하나씩(즉 10개의 격자점을 동시에) 차례로 업데이트한 후 자신의 경계치(굵은 수직선상의 격자점의 값)를 인접 프로세스에게 전달하고 또한 인접 프로세스가 업데이트한 경계치의 값을 받아서 다음의 연산을 진행한다. 이 과정은 잔차의 크기가 충분히 작아질 때까지 반복하는데, 각 프로세스가 계산한 부분 해의 잔차에 대한 정보를 어느 하나의 프로세스가 수집하여 수렴여부를 판단하고 계산의 계속 여부를 결정한다.

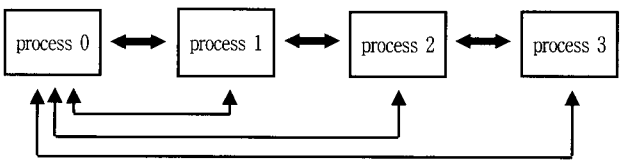
직렬 프로그램과 동일한 결과를 얻기 위해 프로세스들간의 데이터 교환은 동시에 이루어지며, 모든 프로세스가 다음과 같은 SPMD 형태의 동기 알고리즘을 사용할 수 있다[1].

Algorithm 1

1. 병렬 연산 환경을 준비한다.
2. 문제를 풀기 위한 행렬 및 초기치 등을 셋업 한다.
3. 얻어진 해가 만족스러울 때까지 다음을 반복한다.
   1) 담당한 격자선상의 값을 순서대로 1회 업데이트한다.
   2) 좌우측 프로세스에게 자신의 경계치를 전달한다.
   3) 좌우측 프로세스로부터 자신이 필요한 경계치를 수신한다.
   4) 최신 정보를 이용하여 자신의 부분 잔차를 계산한다.
   5) 계산된 부분 잔차의 크기를 process 0에게 보낸다. process 0은 이들 부분잔차를 받아 자신의 것과 더해서 수렴을 판단한다.
   6) 계산의 계속 여부에 대한 지시를 process 0으로부터 받는다.



(그림 1) 20×10의 격자 문제를 4개의 프로세스로 계산하는 예



(그림 2) 4개의 프로세스를 사용하는 경우의 메시지 패싱

이 알고리즘은 3.1)에서 대부분의 시간을 소모한다. 4개의 프로세스는 개념적으로 (그림 2)와 같은 구조로 배열되며, 그 중 process 0이 전체적인 컨트롤을 담당하도록 한다. 프로세스간의 데이터 교환의 대부분을 차지하는 3.2)-3.3)은 굵은 화살표로 표시된 경로로 이루어지며, 전체 연산 진행의 컨트롤에 관련된 정보 교환과정 3.5)-3.6)은 가는 화살표로 표시되어 있다.

3.2 무정지형 해법을 위한 고려 사항

이상의 SPMD 방식에서 어느 프로세스에 장애가 발생하면 이웃한 프로세스들(궁극적으로 모든 프로세스들)은 무한정 기다리게 되어 더 이상의 진행이 불가능하다. 따라서 무정지형 해법을 구현하기 위해서는 다음 세 가지의 문제를 고려해야 한다.

① 한 프로세스의 장애가 다른 프로세스의 연산에 영향을 미치지 않아야 하며
② 메시지 송수신에 있어서 송신함수와 수신함수의 호출이 1:1 매칭이 되지 않아도 되도록 해야하며
③ 어떻게 어느 프로세스의 장애를 파악하여 백업 프로세스를 기동시킬 것인가

3.3 무정지형 해법의 도구

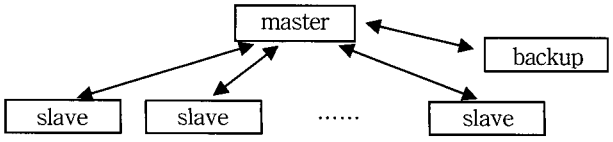
3.3.1 매스터-슬레이브-백업 모델

①의 조건을 만족하기 위해 (그림 3)과 같은 매스터-슬레이브-백업 모델을 사용한다. 즉 슬레이브들은 메시지를 매스터에게만 보내고, 필요한 데이터는 매스터로부터 받도록 한다. 또한 매스터는 각 슬레이브에게 문제가 발생하지 않는지 감시하고, 어느 슬레이브의 장애를 탐지하면 즉시 백업 프로세스에게 장애가 발생한 슬레이브의 번호와 자신이 가진 그 슬레이브가 가장 최근에 보고한 계산치를 전달하여 백업 프로세스가 그 역할을 대신하도록 한다.

어떤 슬레이브의 장애와 무관하게 다른 슬레이브들이 연산을 계속하도록 하려면 비동기 알고리즘을 사용할 수밖에 없다. 또한 비동기 알고리즘은 결과적으로 수렴속도를 높인다.

한편 본 모델에서는 모든 통신이 매스터로 집중되므로 각 슬레이브가 자신이 연산한 모든 중간결과를 매스터로 전송한다면 병목 현상이 염려된다. 예를 들어, 본 논문에서 다루는 문제는 900만×900만이므로, 전체 해 벡터의 길이는 900만이며, 6개의 슬레이브로 나누어 계산할 경우 각 슬레이브가 계산하는 해 벡터의 길이는 150만이다. 따라서 장애 복구에 대비하여 매 연산 단계마다 이를 매스터에게 보낸다면, 배정밀도(double precision) 연산의 경우 데이터 양은 150만×8 bytes×8 bits = 96,000,000 bits가 되며, Gigabit LAN의경우 대략 700-800 Mbps의 실제 전송속도를 가지므로 지연시간(latency)을 제외하고도 순수한 전송시간만 대략 0.15초 정도 걸리며, 실제로는 이보다 훨씬 긴 시간이 소요된다. 이는 각 슬레이브가 1회 반복연산 하는 시간과 비슷하나 매스터는 여러 슬레이브를 상대하므로 슬레이브의 연산 속도에 영향을 미칠 수도 있다.

따라서 각 프로세스는 자신이 계산하는 부분 벡터 전체를 매스터에게 보내지 않고, 자신이 담당한 영역의 경계치만을



(그림 3) 무정지형 매스터-슬레이브-백업 모델

보내고(Algorithm 1처럼), 장애가 발생하여 복구가 필요한 경우, 백업 프로세스는 장애가 발생한 슬레이브가 최종적으로 보낸 양쪽 경계치를 보간법을 사용하여 확장해서 사용하도록 한다. 이 경우, 1회 전송 데이터 양은 좌우 두 경계치를 보내므로 2 × 3000 × 8 bytes×8 bits (3000은 경계치 벡터의 길이)로서 부분 벡터 전체를 보내는 경우에 비해 1/250에 불과하다. 한편 매스터는 실제 연산에 참여하지 않고 데이터의 전송에만 관계하므로 순간적으로 이를 처리할 수 있어서 병목 현상의 염려가 없다.

3.3.2 매스터-슬레이브간의 메시지 송수신

비록 메시지 전달이 슬레이브와 매스터간에만 이루어지더라도, 일반적인 송수신 방법을 사용할 경우 어느 슬레이브에 장애가 발생하면 이로 인해 매스터가 무한정 대기하게 되며, 결과적으로 다른 모든 슬레이브들도 대기하게 된다.

한 가지 해법은, 모든 메시지 전달 요구가 슬레이브들에 의해 시작되며 매스터는 이에 수동적으로 반응을 하도록 하는 것이다. 즉 슬레이브가 데이터를 보낼 경우는 이전과 같으나, 슬레이브가 필요로 하는 데이터는 슬레이브가 매스터에게 요청하여 받도록 한다. 이 경우 매스터는 어느 슬레이브가 데이터를 전송 혹은 요구했는지를 알아야 하는데, 이는 Algorithm 2와 같이 MPI_Iprobe()를 사용하여 보내온 메시지가 있는지 파악하고, 만일 도착한 메시지가 있다면 status라는 구조체를 사용하여 송신자(sender)와 보내온 데이터의 종류(msgkind)를 알아내어 거기에 따라 적절한 행동을 취하도록 한다.

Algorithm 2

```
MPI_Iprobe(MPI_ANY_SOURCE, MPI_ANY_TAG, communic
    ator, &flag, &status) ;
If (flag) {
    sender = status.MPI_SOURCE ;
    msgkind = status.MPI_TAG;
    MPI_Recv( ... );  /* Receive a message from
    sender using msgkind tag. */
    Perform an appropirate job according to the
    message received.
}
```

3.3.3 슬레이브의 장애 파악

매스터는 연산 과정에서 어떤 슬레이브(i 번째라고 하자)가 부분 잔차의 크기를 보내올 때마다 그 시각 $t_{last}(i)$ 및 잔차를 보내오는 평균 시간 간격 $t_{avg}(i)$를 계산하여 기억한다. 만일 i 번째 슬레이브가 시각 $t = t_{last}(i)$에 잔차를 보고했다면, 매스터는 모든 다른 j번째 슬레이브($j \neq i$)들이 다음 조건을 만족하는지를 체크함으로써 이상이 없는지 확인한다.

$$t - t_{last}(j) < \text{TOL} \times t_{avg}(j)$$

TOL은 사용자가 적절히 지정할 허용계수로서, j번째 프로세스의 평균 보고간격 $t_{avg}(j)$의 TOL배가 넘도록 보고하지 않으면 장애를 일으킨 것으로 간주한다.

3.4.4 비동기 반복 알고리즘의 역할

Theorem 1의 의미는, 그 성질을 만족하는 선형 시스템 문제에 비동기 반복 알고리즘을 적용한다면 각 프로세스는 완전히 독립적으로 연산을 수행하여도 항상 옳은 답으로 수렴한다는 것이다. 이는 놀라운 성질로서, 본 논문의 무정지형 알고리즘에서 핵심적 역할을 한다.

즉 Theorem 1의 조건을 만족하는 선형 시스템 문제에 여러 개의 프로세스를 사용한 비동기 병렬 연산을 수행하는 경우, 새로운 무정지 해법의 구현에 있어서

- 각 프로세스들의 계산 진행 상태와 상관없이 비동기적으로 저장한 데이터를 장애 발생시 복구에 사용하여 계산을 재시작해도 되므로 복구 메커니즘을 대폭 단순화할 수 있으며 빠른 복구를 가능하게 한다.
- 어느 프로세스에 장애가 발생하여 복구하는 도중에도 다른 프로세스들은 이와 무관하게 계속 연산을 수행하여도 전체적으로 여전히 옳은 답으로 수렴하게 된다.

이에 비해, 만일 보통의 동기적 알고리즘을 사용한다면 일정 간격으로 모든 프로세스의 상태를 동기적으로 체크포인팅해야 할 것이며, 만일 어느 프로세스에게 장애가 발생할 경우, 모든 프로세스의 상태를 가장 최근의 체크포인트 시점으로 되돌려서 연산을 재시작해야 할 것이므로 복구 과정이 복잡하고 그 기간동안 전체적인 연산의 수렴은 중지될 것이다.

3.4 무정지형 MPMD 알고리즘

이상의 도구들을 사용하여 고안한 슬레이브 프로세스의 알고리즘은 동기적인 Algorithm 1과 거의 동일하나, 데이터 교환을 매스터 프로세스를 상대로 수행한다는 점과, 자신이 필요로 하는 데이터는 매스터에게 요청하여 받도록 한다는 점만이 다르므로 서술을 생략하기로 한다.

매스터 프로세스의 알고리즘은 무정지형 해법 구현의 핵심을 이루며 다음과 같다.

Algorithm 3

1. 병렬 연산 환경을 준비한다.
2. For i=1,2,... until satisfied,
   1) Algorithm 2를 호출하여 슬레이브로부터 온 메시지가 있는지 확인하고
   2) 도착한 메시지가 있으면 발신자(i 번째 프로세스라 하자)와 종류를 파악한다.
   3) 메시지의 종류에 따라 다음을 시행한다.
      i 번째 슬레이브가
      a. 경계치를 보냈다면 이를 수신하고 저장한다.
      b. 인접 슬레이브의 경계치를 요구했다면 그것을 보낸다.
      c. 부분 잔차를 보냈다면
         - 이를 저장하고

- 모든 $j(j \neq i)$번째 슬레이브에 대해, 그 프로세스가 여전히 살아 있는지 점검하고 장애가 감지되면 백업 프로세스를 기동한다.

백업 프로세스는 슬레이브 프로세스의 역할을 대신할 것이므로 당연히 슬레이브의 프로그램과 거의 동일하다. 백업 프로세스는 MPI_Spawn() 함수로 기동시킬 수도 있고, 처음부터 기동하여 대기하도록 할 수 있는데, 본 논문에서는 후자의 방법을 사용하였다. 그 이유는, 장애가 파악되는 즉시 즉각적인 복구가 가능하며, 또한 MPI_Spawn()은 MPI-2에서만 지원이 되므로 이식성이 떨어지기 때문이다. 백업 프로세스의 알고리즘은 다음과 같다.

Algorithm 4

1. 병렬 연산 환경을 준비한다.
2. 매스터 프로세스로부터 지시를 기다린다.
   1) 만일 끝내라는 명령이 오면 시행을 종료한다.
   2) 만일 특정 프로세스의 역할을 대신하라는 명령이 오면 Algorithm 1과 유사한 슬레이브의 알고리즘을 사용하여 연산에 참여한다.

매스터 프로세스는 데이터 전송과 컨트롤에만 관계하며 연산을 담당하는 슬레이브 혹은 백업 프로세스의 프로그램과는 아주 다르므로, 별도의 독립된 프로그램을 사용하는 MPMD (multiple program multiple data) 방식으로 구현할 수 있다.

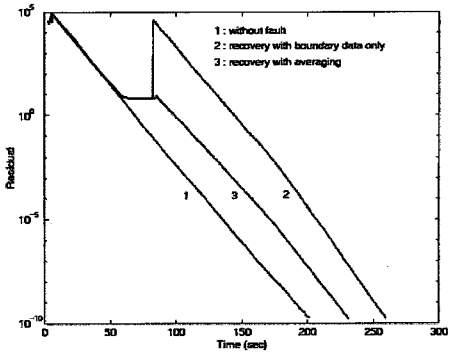
4. 성능 평가

본 실험에서는, 어느 슬레이브에 장애가 발생했을 때 백업 프로세스에 의해 복구되는 데 걸리는 시간과 복구 전후의 수렴을 비교하고자 한다. 성능 평가에는 Gigabit으로 구성된 수원대학교의 Hydra 클러스터를 사용하였는데, 12개의 노드중 8개를 사용(매스터 1, 슬레이브 6, 백업 1)하여 수행하였다.

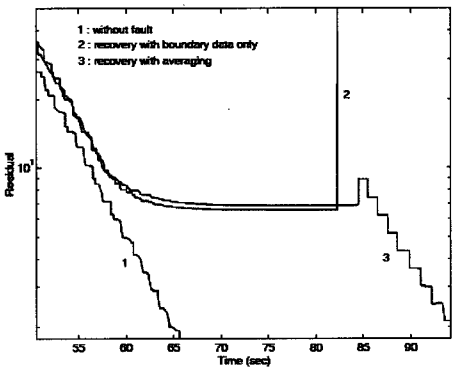
어느 슬레이브의 장애는, 프로그램에서 그 슬레이브를 지정하고 주어진 n_{dead}회의 연산을 수행한 후 프로그램을 종료하도록 함으로써 시뮬레이션 하였다.

(그림 4)는 슬레이브에 장애가 발생하지 않은 경우와 함께, 6개의 슬레이브 중 두 번째 것이 n_{dead}=50회의 연산 이후 장애를 일으켰을 때 TOL=20을 사용하여 이를 감지하고 복구를 할 경우 알고리즘의 수렴을 나타내며, (그림 5)는 일부분을 확대한 것이다.

1은 연산도중 장애가 발생하지 않은 경우를 나타낸다. 2는 백업 프로세스가 계산을 시작할 경우, 단지 매스터로부터 이웃 프로세스가 계산한 최신 경계치만을 받아서 연산을 시작한 경우를 나타내는데, 잔차가 10^4으로 급격히 증가했다가 다시 수렴하기 시작한다. 그 이유는, 비록 이웃 경계의 계산치는 비교적 정확하나 다시 계산을 시작하는 초기치는 이전 계산과는 무관한 값을 가지게 되어 잔차가 급격히 증가하였으며, 이 잘못된 값들이 다른 영역을 계산하는 슬레이브에 나쁜 결과를 미친 것으로 생각된다.



(그림 4) n_{dead}=50의 경우 무정지형 알고리즘의 수렴



(그림 5) (그림 4)의 일부를 확대한 그림

〈표 1〉 n_{dead}=50일 경우, 장애 발생 후 복구에 걸린 시간(초)

n_{dead}	TOL				
	3	5	10	15	20
50	4.80	6.54	9.55	13.39	17.52

반면, 매스터로부터 받은 최신 경계치를 이용하여 x방향으로 1차 보간법으로 구한 값을 초기치로 사용한 경우를 3으로 표시하였다. 이 경우 일시적으로 전체 잔차는 약 1.3배로 증가하나 금방 다시 수렴하기 시작한 것을 보여준다. 2와 3의 경우, 수렴 속도는 장애가 발생하지 않은 경우보다 미약하나마 조금 개선(즉 수렴선이 약간 더 가파르다)된 것을 보여준다.

한편 2와 3의 경우 대략 53초가 경과했을 때 장애가 발생하였으며 이후 복구될 때까지는 5개의 프로세스만이 계산을 계속했음에도 불구하고 거의 65초에 이를 때까지는 계속 수렴했음을 보여주는데, 이는 동기 알고리즘에서는 불가능한 비동기 알고리즘의 특징이다.

<표 1>은 n_{dead}=50일 경우, 어느 슬레이브에 장애가 발생한 후 백업 프로세스가 이를 대신할 때까지 걸린 실제 시간(wall clock time)을 나타낸 것이다. 비록 다양한 TOL 값에 대해 복구에 걸린 시간을 구해보았으나 실제 TOL=3이면 실패없이 장애를 파악함이 밝혀졌다.

FT-MPI와 같은 무정지형 MPI는 어느 노드의 장애가 감지되면 사용자 프로그램으로 하여금 대처할 도구만을 제공할 뿐이며 실제 사용자가 프로그램 내에서 복구 방법을 구

현하여야 한다. 한편 문제가 달라 직접적인 비교는 불가능하나, [6]의 경우 크기가 약 260만인 행렬 문제를 16개의 프로세스로 계산하는 경우 복구에 약 69초(전체 연산 시간의 6%) 소요된 것에 비하면 TOL=3은 아주 짧은 5초 정도(전체 연산 시간의 2.5%)가 소요되었다. 이는 본 알고리즘에서 행렬의 0이 아닌 성분만을 사용함으로써 복구시 백업 프로세스가 비교적 단시간 내에 시작할 수 있으며 장애의 파악 및 복구에 비교적 단순한 방법을 사용하기 때문인 것으로 사료된다.

한편 6개의 슬레이브 중 1, 2, 3번째의 것에 장애가 발생할 경우 각각에 대해 같은 조건으로 실험하여 보았으나, 백업에 걸리는 시간과 알고리즘의 수렴에는 거의 차이가 없는 결과를 얻었다.

5. 결론 및 향후 연구

대규모 병렬 혹은 분산 환경에서, 어느 계산 노드나 통신 네트워크의 장애는 연산 실패를 가져오며, 결과적으로 소중한 계산 시간과 자원이 낭비된다. 그러나 MPI는 이런 문제를 다룰 수 있는 방안을 제시하고 있지 않으며, 대신 여러 무정지형 MPI 라이브러리가 제안되고 있다.

본 논문에서는 무정지형 MPI 라이브러리를 사용하지 않고 기존의 MPI 표준함수만을 사용하여 MPMD 방식으로 비동기 연산을 사용하여 응용 프로그램의 수준에서 무정지형 선형 시스템의 해법을 구현하였다.

성능 실험 결과 제안한 비동기 알고리즘은 동기적 알고리즘에 비해 약간 수렴속도가 빠르며, 장애가 복구된 이후는 장애가 없이 계산이 진행되는 것에 비해 약간 수렴속도가 개선되었다. 한편 하나의 프로세스에 장애가 발생하여 기능을 못하더라도 어느 정도까지는 나머지 프로세스의 연산만으로도 계속 수렴이 진행되는 것이 하나의 특징이다.

그러나 본 논문에서 사용한 비동기 반복 알고리즘은 상당히 많은 부류의 선형 시스템 문제에는 적용이 가능하나 모든 문제에는 적용될 수 없다는 한계도 있다. 한편 (그림 4)의 예에서도 보듯이, 백업 프로세스가 연산을 시작할 때 사용하는 초기치에 따라 연산 시간에 큰 차이가 날 수 있는데, 향후 더 나은 초기치를 얻는 방법에 대한 연구가 필요하다. 또한 거대 문제를 Grid 환경에서 사이트와 사이트 사이에는 제안된 방법을 적용하고 사이트 내에서는 기존의 동기 알고리즘을 사용하는 방법도 고려해 볼 수 있을 것이다.

참 고 문 헌

[1] 박필성, 신순철, "비동기 알고리즘을 이용한 분산 메모리 시스템에서의 초대형 선형 시스템 해법의 성능 향상", 한국정보처리학회 논문지 8-A권, 제4호, pp.439-446, 2000.

[2] R. Bru, V. Migallón, J. Penadés, and D. B. Szyld, "Parallel, synchronous and asynchronous two-stage multisplitting methods," Electronic Transactions on Numerical Analysis, Vol.3, pp.24-38, 1995.

[3] C. Chang, G. Czajkowski, T. von Eicken, and C. Kesselman, "Evaluating the performance limitation of MPMD communication," In Proceedings of SC '97, San Jose, CA, November, pp.15-91, 1997.

[4] D. Chazan and W. Miranker, "Chaotic relaxation," Linear Algebra and Its Applications, Vol.2, pp.199-222, 1969.

[5] R. Cole and Z. Ofer, "An asynchronous parallel algorithm for undirected graph connectivity," TR-546, Dept. of Computer Science, New York University, Feb., 1991.

[6] G. E. Fagg, E. Gabriel, Z. Chen, T. Angskun, G. Bosilca, A. Bukovsky, & J. J. Dongarra, "Fault tolerant communication library and applications for high performance computing," Proceedings of the Los Alamos Computer Science Institute Symposium 2003, Santa Fe, NM., http://icl.cs.utk.edu/news_pub/submissions/lacsi2003-ftmpi-fagg.pdf

[7] MPI Forum. 1995. MPI: A Message-Passing Interface standard.

[8] I. T. Foster, "Designing and building parallel programs," Addison-Wesley Publishing Company, Reading, Massachusetts, 1995.

[9] Frommer, A., Schwandt, H. and Szyld, D. B. (1997). "Asynchronous weighted additive Schwarz methods," Electronic Transactions on Numerical Analysis, vol.5, pp.48-67.

[10] Y. Su and A. Bhaya, "Convergence of pseudocontractions and applications to two-stage and asynchronous multisplitting for singular M-matrices," SIAM J. Matrix Analysis & Applications, Vol.22, pp.948-964, 2001.

[11] D. B. Szyld, "Different models of parallel asynchronous iterations with overlapping blocks," Computational and Applied Mathematics, Vol.17, pp.101-115, 1998.

[12] A. Uresin and M. Dubois, "Parallel asynchronous algorithms for discrete data," Journal of ACM, Vol.37, pp.588-606, 1990.

박 필 성

e-mail : pspark@suwon.ac.kr

1977년 서울대학교 해양학과(학사)

1978년~1982년 KIST 부설 해양연구소 연구원

1984년 미국 올드 도미니언 주립대학 계산학 및 응용수학과(석사)

1991년 미국 매릴랜드 주립대학 응용수학과(박사)

1991년~1995년 한국해양연구소 선임연구원(전산실장, 수치모델 연구그룹장)

1995년~현재 수원대학교 컴퓨터학과(현재 부교수)

관심분야 : 고성능 컴퓨팅, 수치 선형대수, 클러스터 및 분산 컴퓨팅 등