한국통계학회논문집
2012, 19권, 3호, 459–469

DOI: http://dx.doi.org/10.5351/CKSS.2012.19.3.459

병렬처리를 통한 정규혼합분포의 추정

이철희[a], 안성만[1,b]

[a]대교 CNS, [b]국민대학교 경영대학

요 약

본 연구에서는 자기조직화 신경망이 필요한 노드만을 가지고 최적화하여 정규혼합분포를 추정하는 모형 (Ahn과 Kim, 2011)을 Java언어에서 제공하는 스레드(thread)를 기반으로, 멀티코어 컴퓨팅환경에서 병렬처리방식으로 구현하여 순차처리방식에 비해 짧은 연산시간으로 정규혼합모형의 추정이 가능함을 보이려고 한다. 이를 위하여 Ahn과 Kim이 제안한 모형을 바탕으로 두 가지의 병렬처리 방법을 제안하고 그 성능을 평가하였다. 병렬처리 방법은 Java의 멀티스레드를 이용하여 구현되었으며, 모의실험을 통하여 제안한 모형이 순차처리방식과 비교하여 수렴속도가 빠름을 확인하였다.

주요용어: 병렬처리, 정규혼합분포, 수렴속도.

1. 서론

혼합분포는 단일분포(성분)를 결합함으로써 다양한 형태의 확률분포를 나타낼 수 있다. 혼합분포의 성분으로는 정규분포를 많이 사용하고 있으며, 이를 정규혼합모형 (Titterington, 1984)이라고 부른다.

정규혼합모형의 모수를 추정하는 대표적인 방법은 EM알고리듬이며, EM알고리듬의 수렴속도는 많은 경우에 느리다고 알려져 있다 (Dempster 등, 1977). 특히 혼합분포에 EM알고리듬을 적용하는 경우는 결측치가 많이 존재하는 형태이므로 수렴속도가 자연히 느리게 된다. 이 경우에 결측치를 줄여서 수렴속도를 증가시키는 알고리듬이 ECM 알고리듬이며 (Meng과 Rubin, 1991), 이를 바탕으로 CEMM (Celeux 등, 2001), ECME (Liu와 Sun, 1997) 등의 알고리듬이 제안되었다. 한편 Neal과 Hinton (1998)은 혼합분포의 모수를 수정할 때 모든 데이터를 사용하지 않고 일부 데이터만 선택하여 사용하는 방법으로 수렴속도를 높였다.

본 논문에서는 기존의 접근법과 다른 형태의 접근으로서 병렬처리를 통한 수렴속도의 향상을 시도해 보았다. 병렬처리는 복수의 처리 장치를 사용하여, 모든 처리 장치가 하나의 프로그램상의 서로 다른 태스크를 동시에 처리함으로써 처리의 부하를 분담하여 처리 속도를 향상시키는 방법으로서, 성공적인 병렬처리 전환을 위해서는 순차처리모형에 존재하는 종속성을 해석하여 병렬성을 검출하고, 검출된 병렬성을 병렬처리모형으로 전환하는 것이 필요하다 (Kuhn과 Padua, 1981). 각각의 코어에 사용되는 메모리를 분할하여 독립적으로 연산하고, 최종 결과를 갱신 하는 방식을 적용하여 최대한 각 코어 간의 데이터 공유를 줄이고, 이를 통해 시스템의 성능을 증가 시킬 수 있다 (Tian과 Shih, 2009). 즉, 병렬처리의 효과가 커지려면 병렬로 연산되는 단위의 독립성을 높이는 것이 좋다는 것인데, 묶음형태로 실행되는 EM알고리듬의 경우에는 그런 이유에서 병렬처리로 속도향상을 달성하기가 쉽지 않다.

본 연구는 국민대학교 2012년 연구비 지원에 의해 수행되었음.

[1] 교신저자: (136-702) 서울시 성북구 정릉동 861-1, 국민대학교 경영대학, 부교수. E-mail: sahn@kookmin.ac.kr

한편 Yin과 Allinson (2001)은 Kohonen의 SOM모형에 근거한 자기조직화 신경망(SOMN; self-organizing mixture network)으로 정규혼합모형의 모수를 추정하는 방법을 제안하였다. 이 모형에서는 혼합분포의 각 성분이 신경망에서 한 개의 노드로 표현되며 데이터가 신경망에 제시되면 각 노드는 데이터에 대하여 혼합분포에서 한 개의 성분에 대한 모수를 계산하게 된다. 이와 같은 형태의 모형은 개념적으로 병렬처리로 구현되기에 적합하다. 따라서 본 논문에서는 Yin과 Allinson의 모형을 개선한 형태인 Ahn과 Kim (2011)의 모형을 바탕으로 하여 그것을 병렬처리로 구현하여 수렴속도의 개선이 있음을 확인하고자 한다.

본 연구에서는 Ahn과 Kim의 모형을 Java언어에서 제공하는 스레드(혹은 병렬처리의 형태로 동작하는 유닛)를 기반으로, 멀티코어 컴퓨팅환경에서 병렬처리방식으로 구현하여 순차처리방식에 비해 짧은 연산시간으로 정규혼합모형의 추정이 가능함을 보이려고 한다. 그 모형에서는 각 노드에서 한 개의 성분에 해당하는 정규분포의 평균과 분산, 그리고 혼합비율을 추정한다. 그러므로 개념적으로는 각 노드가 자기조직화가 가능하여 쉽게 병렬처리로 구현이 가능하다. 본 연구에서는 두 가지의 병렬처리 방식을 제안을 하는데, 첫 번째 방법에서는 Ahn과 Kim의 모형의 각 노드를 직접 스레드로 변환하는 방식이다. 이 방식은 구현하기가 쉬운 장점이 있지만, 각 노드에서 모수의 값을 개선하는 과정에서 공통적으로 필요한 값이 있어서 그 값이 계산되지 않으면 개별노드의 독립적 실행이 보장되지 못한다. 이러한 메모리의 의존성 문제로 인하여 각 노드를 독립적인 스레드로 구현하여도 속도향상을 얻지 못할 가능성이 있다. 두 번째 방식에서는 마스터 스레드에서 데이터를 가지고서 연산과정의 일부만을 스레드로 실행시키도록 하였다. 이렇게 하면 메모리의 의존성을 피할 수 있게 된다. 그런데 이러한 방식은 연산과정에서 모수가 독립적으로 수정되어서 수렴을 하지 못하거나 반복횟수가 늘어나는 문제가 발생할 수 있다.

본 논문은 다음의 형태로 구성되어 있다. 2절에서는 Ahn과 Kim의 모형을 간략히 소개하며, 3절에서는 본 연구에서 시도한 두 가지의 병렬처리 방식을 설명하고, 각 방식에서 발생할 수 있는 문제점을 논의하며, 4절에서는 모의실험결과를 통하여 두 모형의 성과를 설명하고, 마지막으로 5절에서 요약을 하도록 한다.

2. 신경망을 통한 정규혼합모형의 추정

유한혼합모형의 추정을 위한 신경망은 M개의 입력노드(M은 성분의 수와 같지 않아도 됨)로 구성되며, 각 노드는 하나의 정규분포 핵(Gaussian kernel)을 나타낸다. 핵의 모수는 평균벡터(μ_i)와 공분산행렬(Σ_i)이고 신경망의 출력은 정규분포의 핵을 혼합비율(P_i)로 선형결합한 혼합모형이며 혼합비율도 학습된다. 각 학습단계에서 신경망의 입력값은 표본에서 무작위로 선택되며 각 노드는 다음의 값을 사용하여 학습된다.

$$\hat{P}(i|x) = \frac{\hat{P}_i \hat{p}_i\left(x|\hat{\theta}_i\right)}{\hat{p}\left(x|\hat{\Theta}\right)}. \tag{2.1}$$

식 (2.1)은 EM알고리듬의 E-step에 해당하는 것으로서 데이터값 x를 성분 i가 만들었을 확률을 계산한 것이다. Ahn과 Kim (2011)에서 사용한 모형에서는 신경망의 각 노드에서 모수의 갱신함수를 다음과 같이 표현한다.

$$\hat{\mu}_i(n+1) = \hat{\mu}_i(n) + \alpha(n)\hat{P}(i|x)\left[x(n) - \hat{\mu}_i(n)\right]. \tag{2.2}$$

$$\hat{\Sigma}_i(n+1) = \hat{\Sigma}_i(n) + \alpha(n)\hat{P}(i|x)\left\{\left[x(x) - \hat{\mu}_i(n)\right]\left[x(x) - \hat{\mu}_i(n)\right]^T - \hat{\Sigma}_i(n)\right\}. \tag{2.3}$$

$$\hat{P}_i(n+1) = \hat{P}_i(n) + \alpha(n)\left[\frac{\hat{P}(i|x) - \lambda(1-\alpha)}{1 - g\lambda(1-\alpha)} - \hat{P}_i(n)\right]. \tag{2.4}$$

위에서 $\alpha(n)$은 학습률이고 $0 < \alpha(n) < 1$이며 지속적으로 감소한다. 식에서 λ와 α는 성분을 제거하는 강도를 나타내는 상수로서 $\lambda = 0.0012$ 그리고 $\alpha = 0.01$로 하였다. 식 (2.1)~(2.4)를 사용하여 신경망의 모수를 찾을 수 있으며 결과적으로 정규혼합모형의 확률분포를 추정할 수 있게 된다. Yin과 Allinson (2001)에 의하면 이 모형의 장점은 표본우도를 사용하여 묶음(batch)방식을 사용하는 EM알고리듬에 비하여 수렴속도가 빠르고 표본의 크기가 작아도 수렴하지 못하는 가능성이 낮다.

3. 병렬처리 전환 방법

Molodovan (1986)은 순차 프로그램내의 요소들의 상호관계를 자료 입출력의 선행관계에 의한 자료 종속성 (data dependency)과 실행 시 준수되어야 할 제어흐름 관계에 의한 제어 종속성(control dependency)으로 정의하였고, Angel (1989)과 Taylor (1989)의 연구에서는 병렬처리의 효과를 높이기 위해서는 데이터 의존성을 최소화 할 수 있는 최적의 문제분할 방법과 알고리즘의 병렬성을 높일 수 있는 연구가 매우 중요하다고 주장하였다.

병렬처리 프로그램(parallel program)은 메모리 모델로 분류하면 크게 공유 메모리 모델(thread model)과 분산 메모리 모델(message passing model)로 분류할 수 있다. 공유 메모리 모델의 경우, 복수의 스레드가 실행의 주체가 되고 서로 통신 및 동기화를 맞춰가며 계산이 수행되는 모델을 의미한다. 그러나 스레드 간 실행순서(동기화)를 프로그램이 보증하지 않는 경우에는 원하는 결과를 얻지 못하는 단점이 있다. 분산 메모리 모델에서는 복수의 스레드가 메모리 공간을 독립적으로 가지면서 연산을 수행할 수 있다. 스레드 사이에서 데이터 통신의 필요가 발생하면 송신측(sender)과 수신측(receiver) 사이에서 서로 통신 채널을 연다. 본 연구에서는 공유 메모리 모델을 채용하여 두 가지 방법의 병렬처리 전환 방법을 제안한다.

3.1. 직접 전환법

Ahn과 Kim (2011)의 모형은 연산 전 표본크기, 추정 성분 수, 반복횟수(epoch)를 지정하고, 연산이 진행되면 혼합비율(P)이 작아지는 성분은 탈락이 되면서 실제 분포의 성분 수에 수렴하는 과정을 통해 정규혼합분포의 추정이 이루어진다. 직접 전환법에서는 신경망의 각 노드를 병렬처리 단위로 전환하고 각각의 전환된 노드(앞으로는 병렬처리화 된 연산단위를 '스레드'라 부른다.)는 각 성분의 혼합비율(P), 평균(μ), 분산(Σ)의 값을 갖게 된다. 병렬처리를 하기위해선 표본자료의 공유가 필요하기 때문에, 표본자료를 관리하기 위한 독립적인 스레드가 필요하다. 이 스레드는 각 스레드에서 표본자료를 요청할 때 동일한 표본자료를 제공하고, 각 스레드의 연산을 제어하는 역할을 하게 된다. 이 스레드를 '마스터 스레드'라고 부르겠다.

각 스레드는 마스터 스레드에서 표본을 제공받아 Ahn과 Kim의 모형에 의한 연산을 진행하는 과정을 통해 각 값을 갱신한다. 반복적인 갱신과정을 통해 혼합비율이 낮은 성분은 탈락되기 때문에 반복횟수가 증가할수록 실제 분포의 성분 수에 가까운 스레드만 남게 된다. 시작은 많은 수의 스레드로 시작 되지만 연산이 진행되면서 탈락되는 스레드가 발생하여 보다 빠른 연산이 가능하게 된다. 최종적으로 모든 표본에 대한 연산이 종료되었을 때 남아있는 스레드의 통계치는 이 분포의 추정된 모수가 된다.

이 방법의 장점은 신경망의 각 노드가 직접 병렬처리 단위로 전환이 가능하다는 점이다. 확률분포의 추정을 시작할 때 사용했던 수만큼의 스레드를 생성하여 직접 전환하기 때문에 결과의 정확성(우도를 최대화하는 모수를 추정한다는 의미)과 구현이 용이하다. 반면에 단점은 각 스레드의 연산을 위한 과정에 필요한 일부 값이 공유되어야 한다는 점이다. 각 스레드는 갱신함수 식 (2.2)~(2.4)를 적용하기

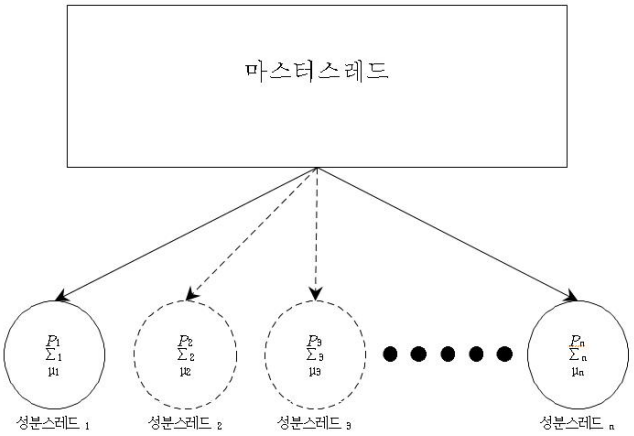

그림 1: *직접 전환법*

위해서 식 (2.1)을 통해 계산된 값이 필요한데, 이 값은 마스터 스레드에서 계산하여 각 스레드에게 전달하는 방식으로 진행된다. 이 과정에서 각 스레드는 다음연산으로 넘어가지 못하고 기다리는 상태가 되고, 만약에 특정 스레드에서 지연이 발생한다면 모든 스레드에 영향을 미쳐 전체적인 성능 저하를 초래할 수 있다. 또한 시작할 때는 성분의 수가 많고 성분의 수만큼 스레드가 필요하게 되므로 그 만큼의 코어가 있는 환경에서는 빠른 처리 시간을 보장하지만, 그보다 작은 환경에서는 순차처리방식보다 못한 성능을 보일 수 있다. 뒤의 모의실험에서 확인할 것이지만 순차방식으로 구현한 경우보다 처리시간이 두 배 이상 빠른 것으로 나타났다.

3.2. 연산 프로세스 전환법

직접 전환법이 각각의 추정 성분을 하나의 스레드로 만드는 병렬처리시스템으로 전환하였다면, 이 방법은 각 표본 하나에 대한 연산을 병렬처리 단위로 전환한 것이다. 각 스레드는 하나의 표본에 대한 모든 성분의 값을 계산하고, 계산된 값을 마스터 스레드에 갱신 한다. 모든 표본에 대한 연산이 끝나면, 최종 추정 통계치가 만들어지는데, 이 과정에서 문제점이 발생할 수 있다. 각 스레드는 실행 속도가 같다고 보장 할 수 없기 때문에, 처리 속도가 빠른 스레드와 느린 스레드가 같이 존재할 수 있다. 각 스레드에 의해 계산된 값으로 갱신하게 되면 다른 스레드에 의해 계산된 값이 무시 될 가능성이 있다. 예를 들어서, 빠른 스레드에 의해서 5개의 표본의 값이 갱신 된 것이, 느린 스레드에 의해서 업데이트 되면서 무시가 될 수 있는 것이다. 따라서 이러한 문제점을 해소하기 위한 보완된 방법이 필요하다.

이를 해결하기 위한 방안으로 각 스레드에서 하나의 표본에 대한 모든 모수의 갱신값만큼 수정하는 방법을 도입하였다. 즉, 각 스레드에서 갱신되는 값을 마스터 스레드에서 관리하고, 모든 표본에 대한 연산이 끝났을 때 마스터 스레드의 값을 통해 이 분포에 대한 최종 추정된 성분을 확인할 수 있다.

각각의 스레드가 메모리를 분할하여 독립적으로 연산하고 최종 결과물을 개선하는 방식을 적용하였기 때문에 직접전환법보다 빠른 수렴속도를 달성할 수 있다. 하지만 일반적인 EM알고리듬과 데이터를 사용하는 방식이 다르기 때문에 추정된 성분의 정확성을 확인하여야 한다. 이 문제는 아래에서 수렴(convergence)에 관해 기술할 때 논의하도록 하겠다.

지금까지 제시한 두 방법을 그림으로 나타내면 그림 1, 그림 2와 같다. 그림에서 점으로 표시된 원은 해당 성분이 실행 중에 탈락될 수 있음을 나타낸다. 각 모형의 병렬처리 과정을 나타내면 그림 3, 그림 4와 같다.

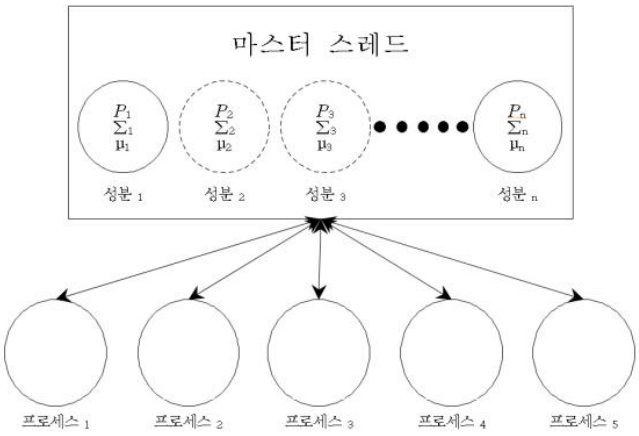

그림 2: *연산 프로세스 전환법*

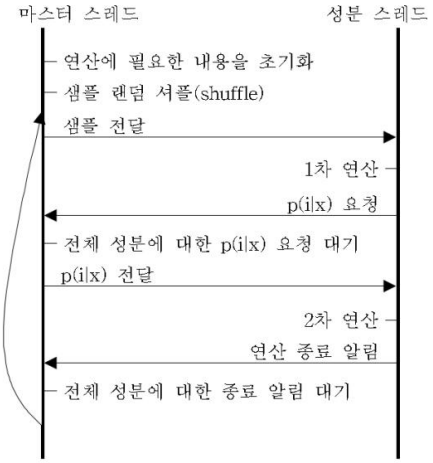

그림 3: *직접 전환법의 처리과정*

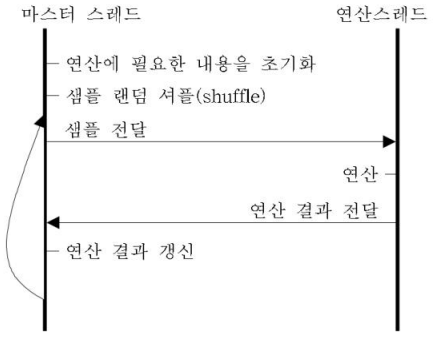

그림 4: *연산 프로세스 전환 모형의 처리과정*

표 1: 실제 분포의 모수

P_1	P_2	P_3	μ_1	μ_2	μ_3	Σ_1	Σ_2	Σ_3
0.345	0.32	0.335	2.5	−1.8	−0.5	4.0 −0.9	3.5 0.75	2.0 0.2
			1.0	2.2	−0.5	−0.9 0.3	0.75 0.3	0.2 0.3

3.3. 병렬처리 알고리듬의 수렴

정규혼합분포를 EM알고리듬과 같은 반복적 방법으로 추정을 하는 경우에 모수의 값이 특정 값으로 수렴한 것을 판단하는 기준으로서, 우도가 더 이상 증가하지 않음을 확인하거나 혹은 모수의 값이 변화하지 않음을 확인하는 두가지 방법이 있다. 본 연구에서는 사용한 모형이 각 모수의 값을 재귀적으로 수정하는 형태이므로 모수의 값의 변화량이 매우 작은 경우에 알고리듬이 수렴한 것으로 간주하였다. 구체적으로는 각 성분의 혼합비(P_i) 변화량과 평균(μ_i) 변화량의 최대값이 기준값(β)보다 작아지면 수렴한 것으로 간주하였으며 식으로 나타내면 다음과 같다.

$$\{\text{MAX}(P(n)\text{변화율}) < \beta\} \quad \text{AND} \quad \{\text{MAX}(\mu(n)\text{변화율}) < \beta\}.$$

위의 식에서 변화율의 최대값은 다음과 같이 정의하였다.

$$\text{MAX}(P(n)\text{변화율}) = \text{MAX}_i \left(\frac{P_i(n) - P_i(n-1)}{P_i(n-1)} \right).$$

$$\text{MAX}(\mu(n)\text{변화율}) = \text{MAX}_i \left(\frac{\mu_i(n) - \mu_i(n-1)}{\mu_i(n-1)} \right).$$

β는 반복적인 실험을 통해 0.005로 설정하였고, 그 경우에 다음절의 모의실험에서 사용한 분포에 대하여 100번의 실험 중 98번이 3개의 성분으로 수렴하는 것을 확인하였다.

수렴에 관련된 또 한가지 논의는, 앞에서 제안한 병렬처리 방법 중 두 번째인 연산프로세스 전환법이 실제 우도를 최대화 하는지에 관한 것이다. 연산프로세스 전환법은 Neal과 Hinton (1998)이 제안한 증분적(incremental) EM알고리듬과 기본적으로 동일하다. 증분적 EM알고리듬은 E-step에서 데이터값 일부를 선택하여 그에 대하여만 모수를 M-step에서 수정하는 방식이며, 데이터값을 선택하는 순서는 임의적이 되어도 무방한 방식이다. 이는 연산프로세스 전환법이 하나의 데이터값에 대하여 모수를 수정하는 점과 데이터값이 선택되는 순서는 Java의 스레드 스케줄러에 의해 임의로 결정된다는 점에서 동일하다고 하겠다.

연산프로세스 전환법이 증분적 EM알고리듬과 다른 점은 선택된 데이터값에 대한 알고리듬의 적용이 병행으로 처리된다는 것인데, 그 때문에 일부 데이터에 대한 모수의 수정값이 다른 데이터에 대한 수정값을 갱신할 때 반영이 되지 못할 가능성이 존재한다. 그렇지만 이 문제는 알고리듬의 수렴속도에 영향을 미칠 수는 있지만, 알고리듬이 우도를 최대화하는 점으로 수렴하는 성질에는 영향을 미치지 않는다. 이와 같은 내용을 다음 절의 모의실험을 통해서 확인할 것이다.

4. 모의실험을 통한 평가

본 연구에서 제안한 병렬처리모형의 성과를 비교하기 위하여 Xu와 Jordan (1996)에서 사용한 2차원 정규혼합분포를 사용해 보겠다. 분포는 세 개의 성분으로 이루어져 있고, 모수의 값은 표 1과 같으며, 분포를 3차원 공간에서 시각적으로 나타내면 그림 5와 같다. 하단의 그림에 있는 원은 성분의 평균(원의 중심)과 혼합비율(원의 상대적 크기)을 나타낸다.

본 연구의 모든 프로그램은 Java로 작성되었고, colt library 1.0버전을 사용하였다. 실험에서 사용한 단일 PC는 Linux OS를 사용하며, 쿼드코어 멀티프로세서를 내장하고 있다. 모의실험은 두가지 형

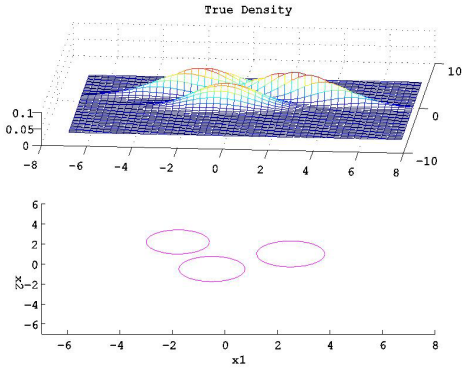

그림 5: *실제* 분포

표 2: 모의실험 측정 결과

모형	최종 성분	개수	반복횟수			평균 소요시간(ms)	1회(epoch) 반복시간(ms)
			최소	최대	평균		
순차	3	185	69	596	177.5	5099.6	
	4	13	151	818	348.6	9427.8	
	5	1	58	58	58	2050.0	
	6	1	61	61	61	2068.0	
	합계	200			187.4	5350.5	28.5
병렬1	3	192	40	463	124.6	1998.9	
	4	8	81	512	207.4	4397.0	
	합계	200			126.7	2094.8	16.5
병렬2	3	191	55	573	195.4	1815.3	
	4	9	171	958	203.8	3948.4	
	합계	200			212.5	1911.3	9.0

태로 진행하였는데, 첫 번째 모의실험은 동일한 조건에서 제안한 병렬처리방법의 성과를 측정할 목적이었고, 두 번째 모의실험에서는 조건을 변경시켜 어떤 병렬처리방법이 특정 조건에서 더 나은 성과를 보이는지 알아보았다.

4.1. 동일한 조건하의 모의실험

모의실험에서는 표본크기를 500, 초기 성분 수를 10, 스레드 수를 5로 설정하였다. 모형의 평가는 연산 시간과 추정된 모형의 정확도를 기준으로 하였다. 연산 시간은 각 모형의 평균 소요시간을 이용해 측정하였고, 모형의 정확성은 우도의 로그값으로 계산하였다. 앞절에서 논의한 바와 같이 병렬처리 알고리듬에 의하여 추정된 모형은 우도를 계산한 결과 대부분의 경우에 유사하게 나왔으므로 실험의 결과의 비교는 연산시간으로 단순화 하였다. 실험의 결과를 표 2에 정리 하였다. 표 2에서 '순차'모형은 Ahn과 Kim (2011)의 모형을 순차적으로 구현한 것으로서 본 연구에서 제안한 병렬처리방법과 비교하기 위하여 포함하였다. '병렬1'은 직접전환법을 나타내며, '병렬2'는 연산프로세스 전환법을 나타낸다.

첫째, 실험을 통해 병렬 처리 방식의 전환을 통해 순차 처리 방식에 비해 향상된 처리 성능을 확인할 수 있었다. 순차 처리 방식은 평균 5.3초의 시간이 소요되었지만, 병렬 처리 방식은 약 2초의 시간이 소요되었다. 병렬처리 전환을 통해 약 2.5배의 효과가 있음이 확인 되었다.

표 3: 성분 수에 따른 모형 별 실행시간 (단위: 초)

	10	12	14	16	18	20
순차	5.23	5.91	7.23	8.02	8.24	9.53
병렬1	2.07	2.08	2.27	2.43	2.45	2.46
병렬2	1.77	2.08	2.18	2.74	3.01	3.10

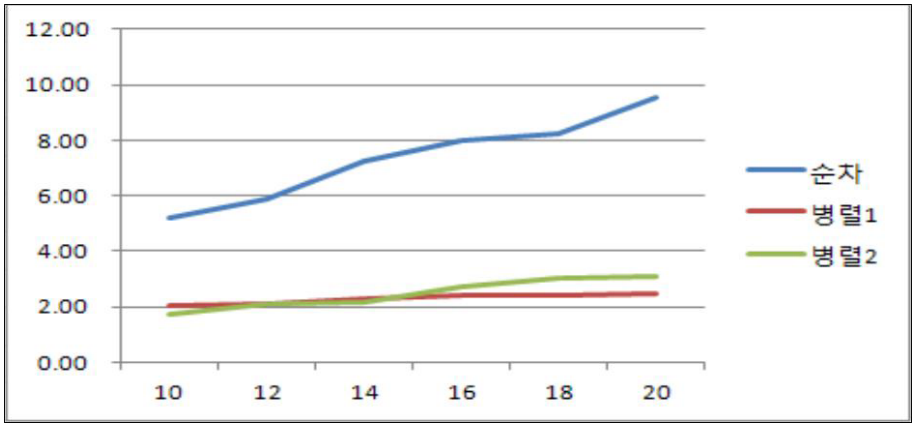

그림 6: *초기 성분 수에 따른 모형별 실행시간*

둘째, 표본에 대한 1회 반복(epoch)시간이 향상되었음을 확인 하였다. 각 모형별로 표본에 대한 1회 반복시간은 평균 28.5(ms), 16.5(ms), 9(ms)로서 병렬처리 전환을 통해 표본을 처리하는 속도가 향상되었음을 알 수 있다.

셋째, 병렬처리 모형1과 모형2 간의 처리 속도의 차이는 크지 않은 것으로 보인다. 표본의 1회 반복시간은 모형2가 빠르지만, 스레드가 전환될 때 다른 스레드에서 변경한 값이 반영되지 못하는 문제로 인해 모형1 보다 더 많은 횟수의 반복을 통해 알고리듬이 수렴하기 때문이다.

넷째, 세 모형의 추정된 성분의 정확성은 차이가 없는 것으로 확인되었다.

4.2. 조건을 달리한 모의실험

앞의 실험을 통해 병렬 처리 전환은 통해 순차 처리 방식에 비해 속도의 개선이 있음을 확인하였다. 한편 병렬처리 알고리듬은 컴퓨팅 환경에 맞게 설정되어야만 최적의 성능을 보장 받을 수 있는데, 여기서는 쿼드코어 프로세서 환경에서 다양한 실험 조건으로 실험을 수행하여 실험 조건의 변화에 따른 각 모형의 성능의 차이를 알아보고, 실험 환경에 맞는 최적의 설정을 제시하고자 한다.

실험 조건은 표본 크기는 500으로 고정 하고, 초기 성분 수와 스레드 수를 변경하였다. 첫 번째 추가 실험은 스레드 수를 고정하고, 초기 성분 수를 10, 12, 14, 16, 18, 20으로 변경하였다. 두 번째 추가 실험은 초기 성분 수를 고정하고, 스레드 수를 1, 2, 3, 4, 5, 6, 7, 8, 9, 10으로 변경하였다. 두 실험을 통해 각 모형의 처리 시간을 비교, 평가하였다.

표 3과 그림 6에는 모형별 전체 소요 시간을 정리하였다. 순차모형은 성분 수가 증가함에 따라 실행시간도 증가하는 것으로 확인이 되었고, 병렬1과 병렬2는 초기 성분의 수가 속도에 미치는 영향이 많지는 않지만, 병렬2가 성분의 수에 좀 더 민감한 것으로 확인되었다.

그림 7과 그림 8에는 스레드 수에 따른 모형 별 전체 실행시간과 반복 횟수를 정리 하였다. 병렬2는 최적의 스레드 수를 초과하게 되면 반복 횟수가 급격하게 증가하게 되고 그에 따라 전체 소요

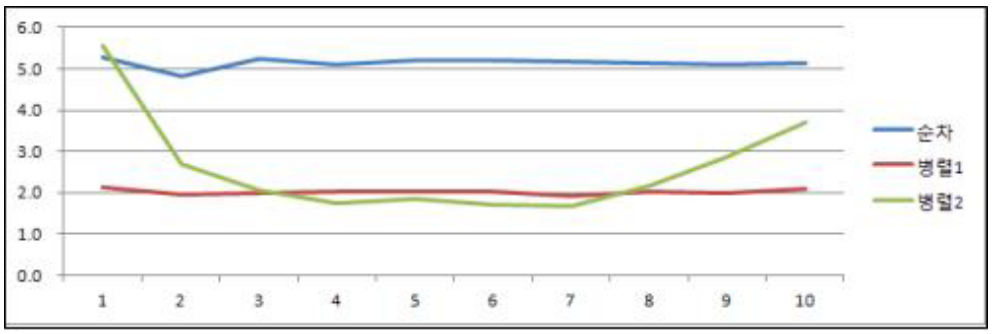

그림 7: *스레드 수에 따른 모형 별 실행시간*

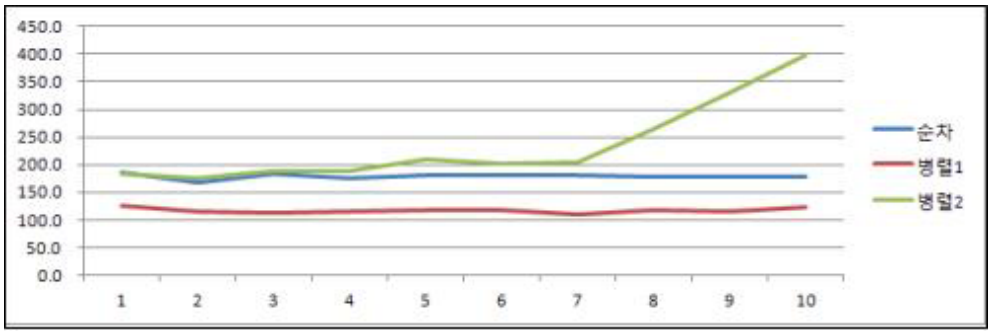

그림 8: *스레드 수에 따른 모형 별 반복 횟수*

시간도 증가하는 것으로 확인되었다. 그 이유는 최대 동시 실행 스레드를 초과 하게 되면, 우선순위에 의해 스레드가 실행되는데, 그 과정에서 지연 현상이 발생하고 표본의 순차적인 처리를 보장할 수 없기 때문에, 자료 종속성에 심각한 문제가 발생하여 많은 반복을 통해 결과에 수렴하는 것으로 판단된다. 따라서 병렬2로 구현하는 경우는 컴퓨팅 환경을 확인하여 코어의 수에 맞는 스레드의 설정이 반드시 필요하다.

5. 요약 및 맺음말

지금까지 정규혼합분포의 추정을 위한 병렬처리 전환 방법을 제시하고 그 성과를 모의실험을 통하여 살펴보았다. 모의실험을 통해 확인한 내용을 요약하면 다음과 같다.

첫째, 병렬 처리 방식의 전환을 통해 순차 처리 방식에 비해 향상된 처리 성능을 확인할 수 있었다.

둘째, 표본에 대한 1회 반복(epoch)시간이 향상되었으며, 특히 병렬2에서 속도가 더 빨랐다.

셋째, 병렬처리 병렬1과 병렬2 간의 수렴속도의 차이는 크지 않았다. 표본의 1회 반복시간은 병렬2가 빠르지만, 스레드가 전환될 때 다른 스레드에서 변경한 값이 반영되지 못하는 문제로 인해 모형1보다 더 많은 횟수의 반복을 통해 알고리듬이 수렴하기 때문이다.

넷째, 세 모형의 추정된 성분의 정확성은 차이가 없는 것으로 확인되었다.

다섯째, 추가 실험에서 스레드 수를 고정하고 초기 성분 수를 다양하게 한 경우에는, 순차적 모형에서는 성분수가 늘어나면 속도가 느려지지만 병렬처리에서는 속도가 상대적으로 영향을 덜 받았다.

여섯째, 추가 실험에서 초기 성분 수를 고정하고 스레드 수를 다양하게 한 경우에는, 병렬2가 코어의 수에 따른 속도의 변화가 심하였다. 따라서 병렬2의 경우에는 코어의 수에 맞는 스레드의 설정이 반

드시 필요하다.

지금까지 병렬처리 전환을 통해 병렬처리 전환에 의해 성능향상이 있을 뿐 아니라 추정된 성분의 정확성에도 문제가 없음을 확인하였다. 병렬처리를 위한 정규혼합분포의 추정은 그 방식이 기존의 속도향상을 위한 ECM과 같은 알고리듬과는 다른 차원에서 접근한 방식으로 멀티코어 컴퓨팅환경을 이용한 속도향상 방법이다. 그러므로 기존의 방법을 대체하는 것이라기보다는 기존의 방법에 보완적으로 적용될 수 있다는 점에서도 의미가 있다고 하겠다.

단일 PC 환경에서의 모의실험을 하였기 때문에, 다양한 멀티코어 환경에서 병렬처리의 성능 향상을 확인 해 볼 수 없는 아쉬움이 있으며, 향후 연구과제로서 단일 PC 환경이 아닌 네트워크 환경과 같은 분산된 컴퓨팅 자원을 이용하여 대규모 연산모델 적용을 검토할 예정이다.

참고 문헌

Ahn, S. and Kim, S. W. (2011). A self-organizing network for normal mixtures, *The Korean Communications in Statistics*, **18**, 837–849.

Angel, L. D. (1989). *The Technology of Parallel Processing*, Prentice-Hall.

Celeux, G., Chretien, S., Forbes, F. and Mkhadri, A. (2001). A component-wise EM algorithm for mixtures, *Journal of Computational and Graphical Statistics*, **10**, 697–712.

Dempster, A., Laird, N. and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm, *Journal of the Royal Statistical Society, Series B*, **39**, 1–38.

Kuhn, R. H. and Padua, D. A. (1981). Tutorial on parallel processing, *IEEE Tenth International Conference on Parallel Processing*.

Liu, C. and Sun, D. X. (1997). Acceleration of EM algorithm for mixture models using ECME, *Proceedings of the Statistical Computing Section, Alexandria, VA: ASA*, 109–114.

Meng, X. L. and Rubin, D. B. (1991). Using EM to obtain asymptotic variance-covariance matrices: The SEM algorithm, *Journal of the American Statistical Association*, **86**, 899–909.

Molodovan, D. I. (1986). Modern Parallel processing, University of Southern California.

Neal, R. M. and Hinton, G. E. (1998). A view of the EM algorithm that justifies incremental, sparse, and other variants, *Learning in Graphical Model*, 355–368 (Jordan, M. I. et al, Eds). Kluwer Academic Press, Norwell, MA.

Taylor, S. (1989). *Parallel Logic Programming Techniques*, Prentice-Hall.

Tian, T. and Shih, C-P. (2009). Software techniques for Shared-Cache Multi-Core Systems, Intel Software Network, http://software.intel.com/ en-us/articles/software-techniques-for-shared-cache-multi-core-systems/

Titterington, D. M. (1984). Recursive parameter estimation using incomplete data, *Journal of the Royal Statistical Society, Series B*, **46**, 257–267.

Xu, L. and Jordan, M. I. (1996). On convergence properties of the EM algorithm for Gaussian mixtures, *Neural Computation*, **8**, 129–151.

Yin, H. and Allinson, N. (2001). Self-organizing mixture networks for probability density estimation, *IEEE Transactions on Neural Network*, **12**, 405–411.

2012년 3월 18일 접수; 2012년 4월 18일 수정; 2012년 4월 20일 채택

Parallel Implementations of the Self-Organizing Network for Normal Mixtures

ChulHee Lee[a], SungMahn Ahn[1,b]

[a]Daekyo CNS; [b]College of Business Administration, Kookmin University

Abstract

This article proposes a couple of parallel implementations of the self-organizing network for normal mixtures. In principle, self-organizing networks should be able to be implemented in a parallel computing environment without issue. However, the network for normal mixtures has inherent problem in being operated parallel in pure sense due to estimating conditional expectations of the mixing proportion in each iteration. This article shows the result of the parallel implementations of the network using Java. According to the results, both of the implementations achieved a faster execution without any performance degradation.

Keywords: Self-organizing network, parallel implementation, normal mixtures.

This work was supported by the research grant of Kookmin University in 2012.

[1] Corresponding author: Associate Professor, College of Business Administration, Kookmin University, 861-1, Jeongneung-dong, Seongbuk-gu, Seoul 136-702, Korea. E-mail: sahn@kookmin.ac.kr