KIPS Tr. Comp. and Comm. Sys.
Vol.4, No.10 pp.337~342 pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2015.4.10.337

Performance Enhancement of Parallel Prime Sieving with Hybrid Programming and Pipeline Scheduling

Seung-yo Ryu† · Dongseung Kim††

ABSTRACT

We develop a new parallelization method for Sieve of Eratosthenes algorithm, which enhances both computation speed and energy efficiency. A pipeline scheduling is included for better load balancing after proper workload partitioning. They run on multicore CPUs with hybrid parallel programming model which uses both message passing and multithreading computation. Experimental results performed on both small scale clusters and a PC with a mobile processor show significant improvement in execution time and energy consumptions.

Keywords : Parallel Computing, Prime Number Generator, Load Balancing, Hybrid Parallel Programming, Message Passing, Multithreading

혼합형 병렬처리 및 파이프라이닝을 활용한 소수 연산 알고리즘

유 승 요† · 김 동 승††

요 약

이 논문에서는 소수 추출 방법인 Sieve of Eratosthenes 알고리즘을 병렬화하여 실행 시간과 에너지 소모 면에서 개선된 효과를 얻고자 실험을 진행하였다. 성능 개선을 위해 부하 균등화를 정교하게 조절하도록 파이프라인 작업 방식을 도입하였고, 멀티코어 컴퓨터 클러스터에 하이브리드 병렬 프로그래밍 모델을 활용하여 효과를 높였다. 소규모 컴퓨터 클러스터와 저전력 컴퓨터에서 구현, 실험한 결과 이전 방식보다 연산 속도가 향상되었고, 에너지 사용량도 감소함을 확인하였다.

키워드 : 병렬 컴퓨팅, 소수 발생기, 부하 균등화, 하이브리드 병렬 프로그래밍, 메시지 전달, 멀티스레딩

1. 서 론

소수(prime number)는 양의 약수가 1과 자기 자신만 존재하는 2 이상의 자연수로 정의된다. 소수는 정수론에서부터 암호학까지의 다양한 분야에서 중요한 역할을 하고 있으며, 특정 범위 안의 모든 소수를 찾는 작업을 소수 발생이라 한다. 대표적인 소수 발생 알고리즘으로 Sieve of Eratosthenes (SOE) 알고리즘이 널리 알려졌다. 큰 소수를 SOE 알고리즘으로 찾는 작업은 문제의 크기가 크고 실행 시간이 길어서 이를 해결하기 위하여 SOE 알고리즘의 병렬화에 대한 연구가 진행되어왔다. 그러나 SOE 알고리즘은 순차적 진행 특성이 강하여, 평이한 병렬화 방법으로는 수행 시간 단축 효과를 크게 얻기 어렵다[1]. 관련 연구로는 SOE 알고리즘의 진행 방식을 수정하고, 작업 분할 방식을 개선하여 성능을 높이거나[2, 3], Wheel Factorization 등의 기법으로 병렬 소수 발생기를 구현한 결과가 보고되었다[4]. 이외에 GPGPU에 적합한 형태로 SOE 알고리즘을 병렬화하는 방안에 대한 선행 연구가 보고되었다[5].

본 논문에서는 SOE 알고리즘의 성능 개선을 위하여 파이프라인 작업 방식을 도입하였다. 또한, 멀티코어 프로세서를 탑재한 병렬/클러스터 컴퓨터에서의 소수 발생기의 성능 향상을 위하여 하이브리드 병렬 프로그래밍 모델을 적용하였다. 하이브리드 모델을 사용하게 됨으로써 데이터 교환 과정이 단순화되고, 병렬 작업 중 쓰레드 간 협력이 효율적으로 이루어져 프로세스와 메시지 전달 방식으로 구현한 이전의 소수 발생기보다 소요 시간이 단축될 것으로 기대된다.

※ 이 논문은 2014년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(NRF-2014R1A1A2055769).
※ 이 논문은 2015년도 한국정보처리학회 춘계학술발표대회에서 '하이브리드 프로그래밍과 파이프라인 작업을 통한 병렬 소수 연산 성능 향상'의 제목으로 발표된 논문을 확장한 것임.
† 준 회 원 : 고려대학교 전기전자공학과 석·박사통합과정
†† 정 회 원 : 고려대학교 전기전자공학부 교수
Manuscript Received : July 3, 2015
First Revision : August 21, 2015
Accepted : August 25, 2015
* Corresponding Author : Dongseung Kim(dkimku@korea.ac.kr)

본 논문의 구성은 다음과 같다. 2절에서는 병렬 프로그래밍 모델에 대하여 설명한다. 3절에서는 SOE 알고리즘과 병렬 작업 분할 방식에 대해 다룰 것이다. 4절에서는 하이브리드 소수 발생기의 구현 방식에 관해 설명할 것이다. 5, 6절은 실험 결과에 대한 검토 분석이 전개되고, 결론을 기술한다.

2. 병렬 프로그래밍 모델

병렬 프로그래밍 모델은 병렬 프로그램을 작성할 때 프로그램이 동작하는 기반을 설명하는 모델이며, 병렬 실행 중인 프로세서들이 상호작용하는 방식을 기준으로 메시지 전달 모델, 공유 메모리 모델, 하이브리드 모델의 3가지로 분류된다[6]. 본 논문에서 사용되는 하이브리드 모델은 메시지 전달 모델과 공유 메모리 모델의 특성을 모두 가지고 있다.

Fig. 1은 하이브리드 모델의 구조이다. 하이브리드 모델에서 다수의 프로세서(Processing Element, PE)들은 그룹을 이루고 같은 그룹에 속한 프로세서들은 공유 메모리를 통해 상호 간의 자료 교환 및 협력을 할 수 있다. 한편 그룹과 그룹 사이에는 상호연결망(Interconnection Network)을 통해 일정한 형식의 메시지를 송수신하는 것으로 병렬 작업에 필요한 정보를 주고받는다. 하이브리드 모델은 공유 메모리 영역의 접근 제어와 동기화, 메시지 송수신 등을 고려해야 하는 어려움이 있으나, 같은 그룹 내의 프로세서 간의 통신이 메모리 읽기, 쓰기로 대체되어 메시지 전달 방식보다 통신량이 적고, 순수한 공유 메모리 방식보다 확장성이 좋다.

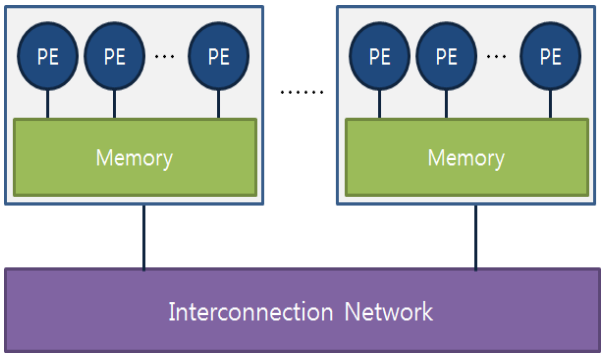

Fig. 1. Structure of Hybrid Parallel Programming Model

3. SOE 알고리즘과 작업 분할

3.1 SOE 알고리즘

SOE 알고리즘은 N 이하의 양의 정수 중 소수만을 추출해 출력하는 소수 발생 알고리즘이다[1, 7]. 이 방법은 A[2:N]의 모든 원소를 0으로 초기화한 후, 임의의 i가 비소수로 판별되면 A[i]=1로 표시하도록 하는데, 최초의 소수인 2부터 차례로 소수 각각의 정수배가 되는 비소수 i를 제거하는 과정을 반복하여 마지막까지 제거되지 않은 모든 값(A[j]=0인 모든 j 값)을 소수로 판정하는 방식이다. 부언하면, 2에 대한 배수를 제거한 후에는 2 이후의 값 중 (제거되지 않은) 최초의 수인 3을 그다음 소수로 확정하고, 같은 방식으로 3의 배수를 모두 제거한다. 이 과정을 새로 확정된 소수가 $\sqrt{N}$ 이상일 때까지 반복한다. 이때까지 남은 값이 2~N 구간 내에 존재하는 전체 소수들의 집합이다. 한편, 2~N 구간 내의 비소수를 제거하는 데 활용된 소수들을 "*sieving prime set* 혹은 *sieving primes*"(이하 SP로 표기함)라 칭한다.

SOE 알고리즘의 동작을 분석하면, **SP**에 포함시킬 새로운 소수 p는 이미 **SP**에 속한 모든 소수를 이용하여 비소수를 제거한 후에야 얻어진다. 이것이 SOE 알고리즘 내부에 존재하는 순차성이다. 따라서 다음 회차의 비소수 제거(sieving)에 사용될 **SP**는 현재 비소수 제거 작업이 종료되기 전까지 알 수 없다.

앞서 다룬 의존성에 의해, 병렬화된 SOE 알고리즘의 동작은 **SP**를 모든 프로세서에 배포한 후, 병렬로 비소수 제거 작업을 진행하고, 다음에 사용할 SP를 찾는 과정을 반복하는 것으로 구성된다. 이 병렬화 방식은 잦은 전역 통신과 동기화가 발생하여 병렬 수행의 단위가 작아져 병렬화의 효과가 떨어진다.

본 논문에서는 SOE의 병렬화 효과를 개선하기 위하여 다음과 같이 수정된 알고리즘을 이용한다[2]. 먼저, 기존의 SOE 방식으로 2~ $\sqrt{N}$ 사이의 소수들을 구한다. 그 후, 얻어진 소수들을 SP로 사용하여 $\sqrt{N}$ ~N 구간의 비소수 제거를 진행한다. 수정된 SOE는 알고리즘의 의존성이 존재하는 부분을 분리하였기 때문에 병렬화하면 기존 방식보다 병렬 수행의 단위가 크고, 병렬화의 효과가 크다.

이 병렬 SOE 알고리즘은 의존성이 존재하는 구간(2~ $\sqrt{N}$)의 소수를 모두 찾아 **SP**를 구하고, 그것을 바탕으로 투입된 전체 프로세서가 동시에 비소수 제거 작업을 진행한다. 이를 위해 프로세서별로 비소수 제거를 담당할 수의 범위를 정해 배열 A를 분할해야 하고, **SP** 또한 나누어 각 프로세서에 배치해야 한다. 이것을 구현한 병렬 알고리즘이 Fig. 2에 보인 par_prime_gen이다. par_prime_gen은 **SP**를 구하는 순차 실행 부분, 각 프로세서별로 SP와 제거 범위를 분할하는 부분, 본격적으로 병렬로 제거 작업을 진행하는 (의존성이 없는) 부분으로 구성된다.

3.2 병렬 소수 발생기의 작업 분할

병렬 소수 발생기의 작업 분할 방식은 소수 여부를 기록하는 배열 A와 $\sqrt{N}$ 이하의 **SP**를 저장하는 배열 S를 각 프로세서에 분할, 배치하되 모든 프로세서의 계산시간이 같아지도록 나누는 것을 목표로 한다. 기존의 병렬 소수 발생기의 작업 분할 방식은 집합 분할(set partition)과 배열 분할(array partition)이 있다[2]. 본 논문에서는 두 분할 방식의 특성을 결합하여 재구성한 파이프라인 분할(pipeline partition)을 사용하였다[3].

1) 집합 분할

집합 분할 방식은 **SP**를 저장한 배열 S를 나누어 각 작업의 부하를 균등하게 한다. 이는 S를 2부터 $K \equiv \sqrt[4]{N}$ 사이의

Algorithm: par_prime_gen

Input:
A[2:N] - Array for marking.
A[j]=0 for all j=2, 3, ... , N.
Output:
P - Set of prime numbers.
P={ j | A[j]=0 }.
Variables:
id - Processor id.
np - Number of processors.
S - Array of **SP**.
Procedure:
1. **if (id = 0)**
Find prime numbers less than $\sqrt{N}$ and save them into the array **S** in increasing order.
2. Distribute **A[** $\sqrt{N}$**:N]** and **S** to **np** processors with selected partition policy.
3. Each processor sieves **A** by **S,** i.e. each marks for the multiples of **SP.**

Fig. 2. Pseudo Code of par_prime_gen

값과 K부터 $\sqrt{N}$ 사이의 값을 갖는 두 집합으로 나누었을 때, 두 집합을 이용하여 N 이하의 정수에 비소수를 제거하는 작업의 부하를 근사계산한 값이 서로 일치하는 것을 이용한다.

집합 분할 방식을 이용할 경우, 각 프로세서는 전체 N에 대하여 배당된 **SP**로 순차적으로 배수 제거를 진행한다. 작업이 끝나면 각 프로세서는 자신에게 배당된 **SP** 원소의 배수만 제거된 배열 A를 구한 것이므로, 완전한 소수집합 A는 각 프로세서가 개별적으로 구한 A를 reduction(논리연산 OR 연산) 과정을 거쳐야 구해진다. 한편, 이 방식은 과도하게 SP를 분할하게 되면 근사 분할의 오차도가 커져 부하 균등화의 효과가 감소하는 단점이 있다.

2) 배열 분할

배열 분할 방식은 배열 A를 분할하는 작업 분할 방식이다. 이는 $m \equiv N/2$까지의 정수를 $\sqrt{m}$ 이하의 **SP**로 비소수를 제거하는 작업과 m+1부터 N까지의 정수를 $\sqrt{N}$ 이하의 **SP**로 비소수를 제거하는 작업의 부하가 거의 일치함을 이용한다.

배열 분할 방식으로 작업을 분할할 경우 각 프로세서는 S 전체와 할당된 A를 이용하여 비소수 제거 작업을 진행한다. 배열 분할 방식은 작업을 종료하면 소수만 남게 되어 별도의 reduction 연산이 필요 없다. 하지만 병렬성을 높이기 위해 분할규모를 증가시키면 실제 작업 부하와 작업 분할을 위해 근사계산한 값과의 오차가 커져 부하 균등화의 효과가 감소한다.

3) 파이프라인 분할

파이프라인 분할 방식은 집합 분할의 부하 균등화 특성을 보완하여 성능을 개선한 작업 분할 방식이다. 먼저, 프로세서들이 고리(ring)로 네트워크를 형성하고 있다고 가정한다. 각 프로세서는 균일한 크기로 분할된 배열 A를 할당받는다. 또한, 프로세서는 집합 분할 방식과 같이 **SP**의 배열 S를 할당받는다. 작업 영역이 할당된 후, 각 프로세서는 비소수 제거를 수행한다. 할당된 작업이 종료되면, 각 프로세서는 자신에게 할당되었던 A를 다음 프로세서로 전달하고, 자신은 다른 프로세서로부터 다른 A의 일부를 전달받아 비소수 제거를 수행한다. 이 과정을 반복하여 모든 A에 대해 비소수 제거를 수행한다. 이때, A의 일부를 전송하는 과정을 반복하지만, reduction 연산은 수행하지 않는다. 파이프라인 방식은 A와 S의 일부를 이용한 비소수 제거를 단위 작업으로 분배한다. 이는 배열 분할이나 집합 분할 방식의 단위 작업보다 규모가 작으므로 작업 균등 배분에 더 적합하다. 하지만 파이프라인 방식은 모든 프로세서가 할당된 작업을 완료해야 다른 프로세서로 배열 A를 전달하고 다음 작업을 시작할 수 있다는 단점이 있다(Fig. 3 참조).

파이프라인 방식을 공유 메모리 모델에 구현할 경우, 배열 A와 S를 공유 메모리 영역에 배치하여 CPU 내의 이웃 코어/프로세스가 직접 접근할 수 있도록 한다. 즉, 다른 프로세서로 배열을 전송하는 대신 프로세서별로 해당 배열의 포인터 값만 갱신하도록 하여 통신 시간을 줄일 수 있다. 또한, 배열의 중복 저장을 피하여 메모리 사용량이 적으며, 디스크 I/O 작업이 중복되지 않아 경합을 줄일 수 있다. 하지만 모든 프로세서가 작업을 종료한 후에 포인터 값을 갱신해야 공유 메모리 영역의 중복 접근을 방지할 수 있으므로 동기화 함수를 삽입해야 한다. Fig. 3 공유 메모리 환경의 쿼드코어 컴퓨터에서 파이프라인 분할 방식의 작업 흐름을 보인다. P_0부터 P_3의 4개의 코어가 작업을 수행하며, 공유 메모리 공간의 배열 S와 A는 S_0~S_3, A_0~A_3로 분할되어있다. t_0~t_3는 각 비소수 제거(sieve) 작업의 시작 시각이다. 먼저 t_0에 각 코어는 주어진 영역에 대해 sieve를 시작한다. 이후 t_1부터 t_3까지 각 코어는 자신이 작업할 A의 영역을 순차적으로 갱신하면서 sieve를 수행한다.

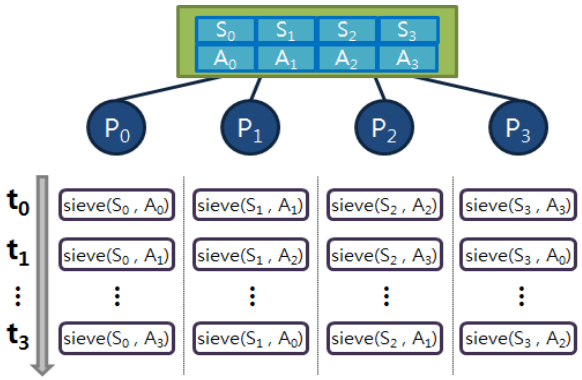

Fig. 3. Workflow of Pipeline Scheduling in Quad-core Shared Memory System

4. 하이브리드 소수 발생기의 구현

하이브리드 연산모델은 클러스터 구성원 CPU들 간에 MPI 통신[8]을 지원하고, OpenMP[9]로 각 멀티코어 컴퓨터 내부에서 메모리 공유가 이뤄지도록 구현하였다. 따라서 프로세서 그룹은 MPI 프로세스와 대응되고, 그룹 내의 프로세서는 OpenMP 쓰레드에 대응된다.

소수 구간(N)이 충분히 큰 경우, 배열 A와 S는 각각 파일로 저장된다. A의 원소를 저장하는 단위를 작게 할 경우, A가 차지하는 용량은 적어지나 비소수 제거 작업이 복잡해진다. 따라서 작업 속도를 유지하면서도 배열의 용량을 최소화할 수 있도록 A의 원소의 크기를 1byte로 사용하였다.

n개의 MPI 프로세스와 프로세스마다 m개의 OpenMP 쓰레드를 사용한 병렬 소수 발생기의 전체 흐름은 Fig. 4와 같다. (1) 먼저 루트 프로세스 P0가 *sieving prime set* S를 생성하고 모든 프로세스로 broadcast 한다. (2) 다음으로 각 프로세스는 배열 분할 방식에 의하여 배열 A의 일부를 할당받게 된다. 프로세스 간의 작업 분할 방식으로 배열 분할 방식을 사용하여 sieving 작업 이후 reduction 연산 등의 후처리 작업을 생략하였다. (3) A를 할당받은 프로세스는 다시 m개의 쓰레드로 분기되고, (4) 각 쓰레드는 파이프라인 방식으로 할당된 비소수 제거 작업(pipeline_sieve)을 수행하게 된다. 쓰레드의 비소수 제거 작업은 공유 메모리를 이용한 파이프라인 방식의 작업 분할로 더 나은 성능을 기대할 수 있다.

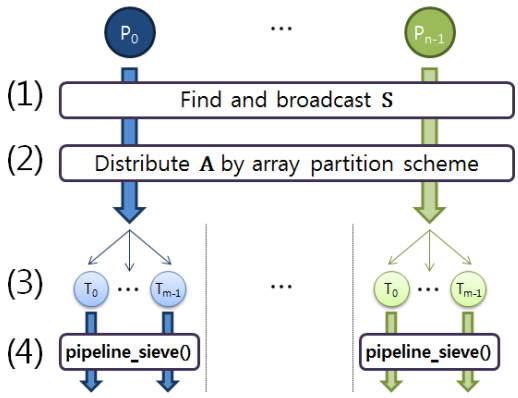

Fig. 4. Workflow of Hybrid Prime Generator

5. 실험 및 결과

병렬 SOE 알고리즘을 실험한 환경은 Table 1에 정리하였다. 실험은 세 종류의 환경에서 진행되었으며, 각각 Intel E7200 CPU를 장착한 4대의 PC로 구축된 E7200 클러스터, Intel i7-3770 CPU를 장착한 2대의 PC로 구축된 i7-3770 클러스터, 그리고 저전력 CPU인 Intel N2930을 장착한 PC 한 대이다. 컴퓨터 클러스터는 기가비트 이더넷 스위치를 이용하여 네트워크를 구축하였다. 각 실험에서 소수를 찾는 최대 범위는 N으로, 사용한 프로세서 코어의 수는 N_P로 표기하였다.

Table 1. List of Experimental System

CPU	Intel E7200	Intel i7-3770	Intel N2930
Clock speed	2.53GHz	3.40GHz	1.83GHz
Number of cores	2	4	4
Cache size	3MB	8MB	2MB
RAM size	8GB	8GB	4GB
HDD capacity	500GB	500GB	500GB
Configuration	Cluster of 4 PCs	Cluster of 2 PCs	Single PC

먼저 병렬 SOE 알고리즘의 집합 분할 방식과 배열 분할 방식을 구현하여 E7200 클러스터에서 실험하였다. 실험에서는 분산메모리 방식의 메시지 전달 모델에 SOE 알고리즘을 적용하였다. Fig. 5는 SOE 알고리즘의 실행 시간을 N과 N_P를 변화시키면서 측정한 결과로, N 및 2N(단, $N=10^{10}$)에 대해 실험하였다. 그 결과 배열 분할 방식이 집합 분할 방식에 비해 우수한 것으로 관측되었는데, 배열 분할 방식은 프로세서 코어의 수(N_P)가 증가함에 따라 실행 시간이 감소하는 반면, 집합 분할 방식은 그렇지 못했다. 그 원인은 Fig. 6에서 볼 때, 배열 분할 방식의 sieve 작업 후에 실행되는 reduction 연산의 작업량 증가인 것으로 추정된다.

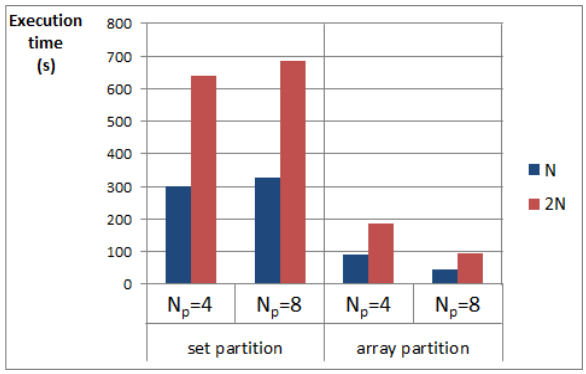

Fig. 5. Execution Time of Two Load Balancing Schemes in Parallel Prime Generators

Fig. 6은 집합 분할 방식에 사용한 프로세서 코어 수 증가에 따른 sieving과 reduction 연산의 실행 시간의 그래프이다. sieving과 reduction의 실행 시간의 합이 SOE의 전체 실행 시간과 거의 일치하였으며, 이를 통해 실행 시간에 영향을 준 요인을 확인하였다. 실험 결과, 병렬 실행에 N_P를 증가시킴에 따라 sieve 연산의 실행 시간은 감소하고, reduction 연산의 실행 시간은 증가하는 것을 확인할 수 있다. sieve 연산의 성능 개선 효과는 N_P가 증가함에 따라 감소하는 경향을 보였으며, 집합 분할 방식의 실제 sieving 작업의 균등 분할 성능이 떨어짐을 알 수 있다. 한편 reduction 연산은 N_P가 증가할수록 실행 시간이 증가한다. 이는 코어 수가 증가하면 각 코어의 배열 A의 크기는 변하지 않지만, reduction에 사용되는 전체 데이터의 크기는 (배열 A의 크기 × 프로세서

코어의 수)이므로 reduction 연산의 작업량이 증가하기 때문이다. 결과적으로 사용한 프로세서 코어의 수가 증가함에 따라 sieve 연산의 성능 개선 효과보다 reduction에 의한 소요 시간 증가가 우세해져 역효과를 낳게 된다.

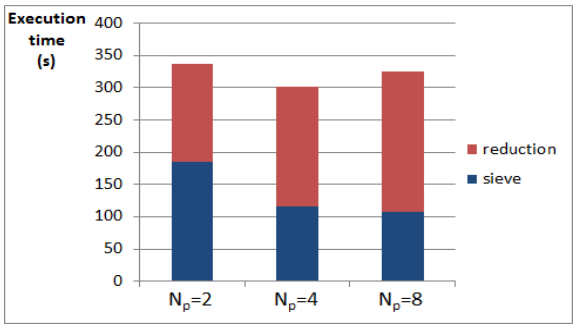

Fig. 6. Execution Time of Sieving and Reduction Operation in SOE Algorithm with Set Partition Scheme when $N=10^{10}$

Table 2는 배열 분할 방식을 적용하였을 때 배치된 부하량을 가장 작업량이 많은 코어에 대한 코어별 작업량의 비로 표시한 표이다. 코어의 부하량은 실행 시간을 기준으로 (코어의 실행 시간 ÷ 코어의 실행 시간의 최댓값)으로 계산하였다. 따라서 작업량이 가장 많은 코어의 작업량의 비는 1.0이며, 다른 코어들은 1.0 이하의 값을 가진다. 실험 결과를 통하여 N_p가 작을수록 코어별 작업량이 균등한 것을 확인할 수 있다. 이는 코어의 수가 증가하면 분배의 단위 작업의 수량이 상대적으로 부족함에 기인한다. 반면 N의 크기는 작업량의 비의 변화에 큰 영향을 주지는 못함을 확인할 수 있다.

Table 2. Workload Ratio for Each Processor when Using Array Partition Scheme with $N=10^{10}$

	Np=4		Np=8	
	N	2N	N	2N
core 0	0.92	0.93	0.89	0.90
core 1	0.95	0.95	0.92	0.92
core 2	0.99	1.00	0.95	0.96
core 3	1.00	1.00	0.97	0.97
core 4			0.98	0.98
core 5			0.99	0.99
core 6			0.99	0.99
core 7			1.00	1.00

다음 실험은 하이브리드 소수 발생기의 실행 시간과 에너지 소모량을 측정하고, 그 성능을 기존 소수 발생기와 비교하였다. 비교 대상인 기존 소수 발생기는 앞선 실험에 사용된 메시지 전달 모델 기반의 배열 분할 방식 소수 발생기이다. 실행 시간은 막대로, 에너지 소모량은 선으로 그래프에 표시하였다.

Fig. 7은 Table 1의 세 가지 환경에서 두 소수 발생기의 실행 시간과 에너지 소모량을 나타낸 그래프이다. 실험에서 $N=4\times10^{10}$으로 고정하였다. 먼저 E7200 클러스터에서는 Np=2인 경우를 제외하면 하이브리드 소수 발생기의 성능이 기존 소수 발생기의 성능보다 약간 떨어졌다. 그 이유는 E7200 프로세서는 코어 개수가 너무 작아 작업 분할이 상대적으로 균등하지 않았던 것으로 추정된다. i7-3770과 N2930 프로세서는 모두 4개의 코어로 구성되어 있고 이들의 경우 모두 하이브리드 방식이 더 빠르게 작업을 수행하였다. 이는 병렬성이 높은 공유 메모리 환경의 파이프라인 작업 분할 방식이 배열 분할보다 부하 균등화가 정밀한 것으로 볼 수 있다. 한편 i7-3770에서는 기존 방식과 하이브리드 방식의 에너지 소모량이 거의 비슷했지만 N2930에서는 하이브리드 방식에서 큰 폭의 실행 시간 개선 및 에너지 소모량 감축이 있었다. 이는 모바일 프로세서의 경우 하드웨어 성능이 부족하여 기존 소수 발생기의 다량의 메모리, 디스크 접근을 소화하지 못했지만, 하이브리드 방식은 데이터 중복 저장을 피하고 가벼운 쓰레드를 사용하여 에너지 소모량을 줄인 것으로 보인다.

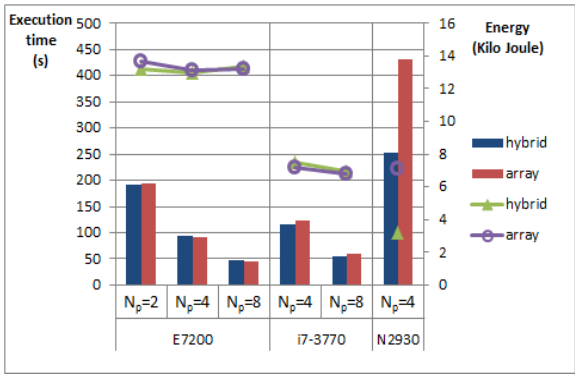

Fig. 7. Execution Time and Energy Consumption of 2 Parallel Prime Generators with $N=4\times10^{10}$

Fig. 8은 i7-3770 클러스터에서 $N=10^{10}$과 2N, 4N에 대하여 두 소수 발생기의 실행 시간과 에너지 소모량을 나타낸 그래프이다. 이전 결과와 달리 N이 증가했음에도 하이브리드 소수 발생기는 기존 방식보다 최대 7%의 성능 개선을 보이며, 에너지 소모량은 기존 방식과 비슷하거나 조금 많았다. 이는 하이브리드 방식이 작업 분할을 개선하여 더 빠른 작업 수행을 하면서도 에너지 소모가 증가하지 않음을 보여주는 것이다.

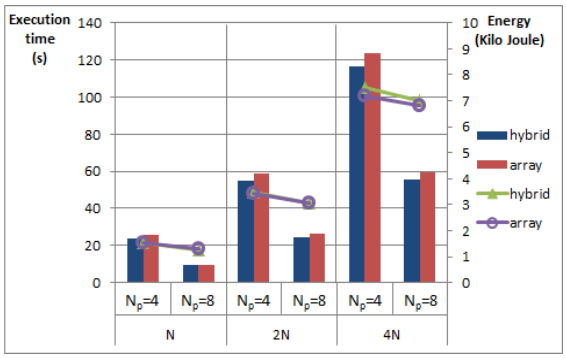

Fig. 8. Execution Time and Energy Consumption of 2 Parallel Prime Generators in i7-3770 Cluster

6. 결 론

본 연구는 소수생성 SOE 알고리즘의 순차적인 진행 속성을 위배하지 않고, 병렬화 시에 직면하는 부하 불균형 문제를 해결하는 것을 목표로 하였다. 이를 위하여, 전체 소수 계산 구간을 블록 단위로 분할하여 프로세서에 공급하고, 각 프로세서가 sieving을 진행한 후, 그 결과물(블록)을 다음 프로세서로 넘겨주는 파이프라인 구조를 적용하였다. 또한, 멀티쓰레딩과 메시지 교환을 병행하는 하이브리드 프로그래밍을 통해 멀티코어 CPU의 특성을 최대한 활용하여 성능 향상을 노렸다. 실험 결과, 기존 방식보다 현저하게 실행 시간을 단축할 수 있었다. 고안된 방식은 순차성이 강하고, 균등한 부하 분산이 어려운 유사 알고리즘의 병렬화 문제를 해결하는 데 기여할 것이다.

References

[1] S. H. Bokhari, "Multiprocessing the sieve of Eratosthenes," *IEEE Computer*, Vol.20, No.4, pp.50-58, 1987.

[2] S. Hwang, K. Chung, and D. Kim, "Load balanced parallel prime number generator with Sieve of Eratosthenes on cluster computers," in *Proceedings of the IEEE International Conference on Computer and Information Technology*, Fukushima, pp.295-299, 2007.

[3] S. Hwang, "Parallel prime number generator on cluster computers," Master dissertation, Korea University, Seoul, Korea, 2008.

[4] C. M. Costa, A. M. Sampaio, and J. G. Barbosa, "Distributed prime sieve in heterogeneous computer clusters," in *Computational Science and Its Applications-ICCSA 2014*, Springer International Publishing, pp.592-606, 2014.

[5] G. Bhat, N. G. Kini, R. Siddharth, S. Prabhu, and G. Hegde, "Parallelization of sieve of Eratosthenes," *International Journal of Scientific Research in Computer Science Applications and Management Studies*, Vol.3, No.5, 2014.

[6] B. Chapman, G. Jost, and R. van der Pas, *Using OpenMP: portable shared memory parallel programming*, MIT press, 2008.

[7] X. Luo, "A practical sieve algorithm for finding prime numbers," *Communications of the ACM*, Vol.32, No.3, pp. 344-346, 1989.

[8] The Message Passing Interface (MPI) standard [Internet], http://www.mcs.anl.gov/mpi.

[9] The OpenMP® API specification for parallel programming [Internet], http://openmp.org/wp.

유 승 요

e-mail : gpgun@korea.ac.kr

2013년 고려대학교 전기전자전파공학부(학사)

현 재 고려대학교 전기전자공학과 석·박사통합과정

관심분야 : 고성능 컴퓨팅, 병렬 알고리즘, 빅데이터

김 동 승

e-mail : dkimku@korea.ac.kr

1978년 서울대학교 전자공학과(학사)

1980년 한국과학기술원 전자전기공학과 (석사)

1980년~1983년 경북대학교 전임강사

1988년 University of Southern California(박사)

1988년~1989년 University of Southern California, postdoc

1989년~1995년 포항공과대학 부교수

1995년~현 재 고려대학교 전기전자공학부 교수

관심분야 : 고성능 컴퓨팅, 병렬 알고리즘, 빅데이터