JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 27 No. 2, pp. 1-8, February 2022
https://doi.org/10.9708/jksci.2022.27.02.001

GPU-based Parallel Ant Colony System for Traveling Salesman Problem

Yunseok Rhee*

*Professor, Division of Computer Engineering, Hankuk University of Foreign Studies, Yongin, Korea

[Abstract]

In this paper, we design and implement a GPU-based parallel algorithm to effectively solve the traveling salesman problem through an ant color system. The repetition process of generating hundreds or thousands of tours simultaneously in TSP utilizes GPU's task-level parallelism, and the update process of pheromone trails data actively exploits data parallelism by 32x32 thread blocks. In particular, through simultaneous memory access of multiple threads, the coalesced accesses on continuous memory addresses and concurrent accesses on shared memory are supported. This experiment used 127 to 1002 city data provided by TSPLIB, and compared the performance of sequential and parallel algorithms by using Intel Core i9-9900K CPU and Nvidia Titan RTX system. Performance improvement by GPU parallelization shows speedup of about 10.13 to 11.37 times.

▸ **Key words**: Ant colony system, Traveling salesman problem, Graphic processing unit, Metaheuristic, Parallelization

[요 약]

본 논문에서는 개미 집단 시스템(ant colony system)을 통한 순회 외판원 문제(traveling salesman problem)를 효과적으로 해결하기 위해 GPU 기반 병렬 알고리즘을 설계 구현하였다. TSP에서 동시에 수백 또는 수천의 탐색 여정(tour)을 생성하는 반복 과정을 GPU의 작업 병렬성을 활용하여 처리성능을 개선하고, 페로몬 자취 데이터의 업데이트 과정은 32x32의 쓰레드 블럭을 사용하여 데이터 병렬성을 적극 활용하였다. 특히 다중 쓰레드의 메모리 동시 접근을 통해 연속 메모리공간의 병합 접근 효과와 공유 메모리의 동시 접근을 지원하였다. 본 실험은 TSPLIB에서 제공되는 127개부터 1002개에 이르는 도시 데이터를 사용하였고, Intel Core i9-9900K CPU와 Nvidia Titan RTX 시스템을 사용하여 순차 알고리즘과 병렬 알고리즘의 성능을 비교하였다. GPU 병렬화에 의한 성능 향상은 약 10.13~11.37배의 성능 개선 효과를 보였다.

▸ **주제어**: 개미 집단 시스템, 순회 외판원 문제, 그래픽 처리 장치, 메타 휴리스틱, 병렬화

• First Author: Yunseok Rhee, Corresponding Author: Yunseok Rhee
*Yunseok Rhee (rheeys@hufs.ac.kr), Division of Computer Engineering, Hankuk University of Foreign Studies
• Received: 2022. 01. 26, Revised: 2022. 02. 11, Accepted: 2022. 02. 13.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

순회 외판원 문제(traveling salesman problem, TSP)는 여러 도시를 한번씩만 방문하여 일순하는 최적 경로를 구하는 문제로 그 계산복잡도가 NP-hard임이 잘 알려진 문제이다[1]. 이 문제의 해법이 다양한 응용에 사용되는 까닭에 빠른 시간에 최적(또는 근사 최적)의 해를 구하는 방법에 관해 많은 연구들이 수행되었다. 이 가운데 개미 집단 시스템(ant colony system, ACS)는 Dorigo와 Gambardella 등에 의해 제안된 조합 최적화 방법이며, 수많은 조합에서 최적해를 찾아내기 위해 자연계의 특성을 활용한 메타휴리스틱(metaheuristic) 시스템으로 TSP의 근사 최적해(near optimal solution)을 구하는데 효과적인 알고리즘으로 잘 알려져 있다[2]. ACS와 같은 메타휴리스틱 알고리즘은 특정 문제에 국한되지 않고 다양한 문제에 적용할 수 있어서 TSP에 적용한 병렬화 기법과 효과를 유사한 문제에 활용할 수 있을 것으로 기대된다[3].

특히 유전자 알고리즘이나 시뮬레이티드 어닐링(simulated anealing) 기법 등도 TSP의 근사 최적해를 구하는데 많이 활용되고 있지만[4][5], 대부분의 기존 연구들은 문제 해결 과정에서 네트워크 상황이 변하지 않는 정적(static) TSP를 다룬다. 그러나, 현실 상황의 TSP는 도로의 교통량 변화와 같은 동적인 환경을 반영하게 되는데, 차량 라우팅 문제 등에서 같은 동적(dynamic) TSP에 ACS가 매우 효과적인 것으로 알려져 있다[6]-[8].

ACS의 막대한 계산량을 효과적으로 해결하기 위해 다양한 병렬화 연구가 있었다. 대표적으로 다수의 개미 집단을 정의하고 이를 개별 CPU에 할당하거나, 단일 개미 집단을 다중 쓰레드 모델로 구현한 경우이다. 이들은 대개 고속 네트워크로 연결된 다중 컴퓨터 상에서 MPI를 이용해 병렬화하거나 OpenMP를 이용해 다중 프로세서에서 병렬화를 시도했고[9]-[11], 이후 GPU를 이용한 병렬화 연구들이 수행되었다[12]-[14].

GPU 기반 병렬화가 이 문제에 적합한 이유는 네트워크를 구성하는 각 노드에서 인접한 노드에 대해 경로 선택 적합성(fitness)을 계산하는 과정이나 에이전트인 개미(ant)가 방문한 경로에 대해 페로몬(pheromone)의 영향을 업데이트하는 과정들이 모든 노드에서 SIMD 형태의 동일한 계산과정이 GPU의 SIMD 다중 코어에서 효과적으로 수행될 수 있기 때문이다.

2장에서는 ACS 알고리즘을 이용한 TSP 해결 방법과 GPU 프로그래밍 관련 내용을 소개하고 3장에서 본 논문에서 구현한 GPU 기반 병렬 알고리즘을 각 부분별로 상세 기술한다. 그리고 4장에서 CPU 구현 알고리즘 대비 성능 개선 결과와 ACS의 반복 실행 횟수에 따른 결과값의 변화 등을 분석하고 5장에서 결론을 맺는다.

II. Preliminaries

2.1. ACS for solving the TSP

TSP는 ACS 알고리즘이 가장 앞서 다룬 최적화 문제이다[3]. TSP는 주어진 각 도시 $\{v_o, v_1, v_2, \dots, v_{n-1}\}$를 한 번씩만 방문하여 모든 도시를 순회하는 최단 경로를 구하는 문제로서, ACS에서는 개미라고 불리는 다수의 에이전트(agent)들이 다양한 경로를 분산 탐색하는 역할을 한다.

ACS 알고리즘에서는 개미가 이동하면서 각 에지에 페로몬 자취(trail)를 남기는 계산 모델을 사용하며, 이 때 $e_{i,j}$에 남긴 페로몬의 값을 $\tau_{i,j}$로 표현하고 $d_{i,j}$는 두 도시 v_i와 v_j 간의 거리를 나타낸다.

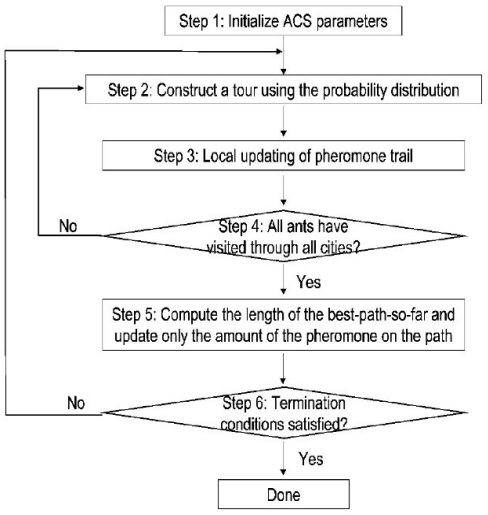

Fig. 1. ACS algorithm flow for solving TSP

그림 1은 ACS 알고리즘을 사용한 TSP 해결 과정을 간략히 보여주고 있다. 우선 Step 1에서는 TSP 데이터를 사용하여 ACS의 개미의 위치, 페로몬 값 등을 초기화한다. 특히 개미의 수는 대개 도시의 수와 같거나 그보다 적은 수로 설정하고, 그 위치는 랜덤하게 배치한다.

Step 2는 각 도시에서 연결된 인접 도시들에 대해 확률적 가능성과 임의 선택(random selection)을 사용하여 다음 방문도시를 선택하고, 모든 도시를 방문할 때까지 이와 같은 과정을 반복하여 하나의 여정(tour)을 생성한다.

ACS에서 다음 도시의 선택 적합도(fitness)는 크게 페로몬의 양($\tau_{i,j}$)에 비례하고 도시 간 거리($d_{i,j}$)에 반비례한

다. 여기에, 이 두 값의 영향력을 통제하기 위해 적당한 양수 α, β를 사용하여 다음과 같이 적합도 $f_{i,j}$, 즉 도시 v_i에서 v_j를 선택하는 확률적 적합도를 구한다.

$$f_{i,j} = (\tau_{i,j})^{\alpha}(1/d_{i,j})^{\beta}$$

또한, 각 인접 도시 v_j에 대해 얻은 $f_{i,j}$를 모두 합한 값 F를 구한 후, 이를 사용하여 다음과 같이 각 도시에 대한 확률을 설정한다.

$$F = \sum_{j} f_{i,j}$$
$$p_{i,j} = f_{i,j}/F$$

앞서 구한 각 확률에 비례한 면적으로 구성된 룰렛 원판(roulette-wheel)에서 임의 위치를 선택함으로써 다음 방문지를 선택한다. 이 방법을 통해 적합도가 높은 도시가 확률적으로 선택 가능성이 높지만 항상 선택되는 것은 아니다.

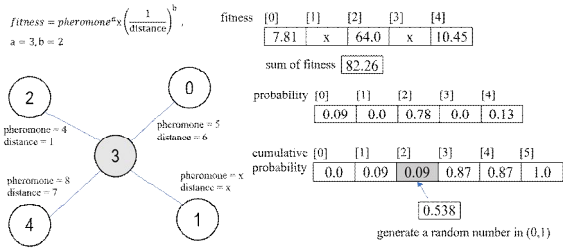

Fig. 2. Stochastic selection for next visiting city

그림 2는 현재의 도시 3에서 다음 방문 도시를 선택하는 과정을 예로 보인다. 3에서 연결된 도시는 0, 2, 4이며 각각의 적합도(fitness)는 7.81, 64.0, 10.45가 계산되고 이들의 합계는 82.26이다. 이 합계에 비례하는 확률(probability)은 각각 0.09, 0.78, 0.13이며, 이들 확률의 누적 확률분포(cumulative probability)를 구한다. 이후 (0,1) 구간에서 발생한 난수 p(0.538)의 값이 cumul[2]와 cumul[3] 사이에 위치하고, 이것은 룰렛 원판에서 도시 2가 무작위 선택된 결과와 같다.

Step 3은 페로몬의 양을 전체적으로 업데이트하는 단계이다. ACS에서는 이미 경로 상에 배출된 페로몬은 시간이 지남에 따라 증발하여 그 양이 다음과 같이 점차 약화된다고 가정한다. 이때 ρ는 실험에 의해 결정된 증발율 상수이다.

$$\tau_{i,j} = (1-\rho)\tau_{i,j}$$

한편, 개미가 지나간 경로에는 페로몬이 증가되는데 그 양의 개미가 이동한 거리에 반비례한다고 가정한다. 따라서, 페로몬의 양이 업데이트되는 식은 다음과 같이 결정된다.

$$\tau_{i,j} = (1-\rho)\tau_{i,j} + 1/d_{i,j}$$

2.2 GPU-based parallel programming

GPU는 데이터 병렬성을 지원하는 아키텍쳐로서 ACS 알고리즘과 같이 에이전트들이 서로 다른 데이터에 대해 동일한 연산을 수행하는 형태의 알고리즘에 적합한 구조이다. 특히, TSP 해결에 필요한 인접 도시의 적합도 판정(fitness calculation), 여정 생성(tour construction), 페로몬 업데이트(pheromone update) 등은 대규모 행렬 데이터에 대한 GPU 병렬처리가 성능의 관건이다.

각 GPU 장치는 다수의 SM(streaming multiprocessor)로 구성되고, 각 SM은 내부에 SIMD 형태로 실행되는 많은 수의 스트림 프로세서(stream processor 또는 core)를 갖는다. 특히 GPU 메모리 시스템은 전역 메모리(global memory)와 공유 메모리(shared memory)의 계층 구조를 갖는데, 전역 메모리는 CUDA kernel의 모든 쓰레드(thread)들이 접근 가능한 메모리이나 매우 느린 반면, 공유 메모리는 각 쓰레드 브럭(thread block) 내의 모든 쓰레드가 공유하는 메모리로 빠른 접근시간을 지원한다.

따라서, 데이터 병렬성이 높은 알고리즘을 GPU에서 구현할 때 전역 메모리 접근 횟수를 줄이고, 공유 메모리를 적극 활용하는 것이 필요하다. 특히 GPU에서 전역 메모리 접근이 차지하는 오버헤드가 매우 크므로 그림 3에 보이는 바와 같이 블록 내의 쓰레드들이 전역 메모리의 연속된 위치에 동시에 접근할 때 GPU에서는 이를 한 번에 처리하여 메모리 접근의 대역폭을 최대한 활용하는 병합 접근(coalesced access)을 지원한다. 이에 반해 일정 간격을 두고 메모리에 접근하는 일정간격 접근(stride access)은 병합 접근을 사용하지 못하므로 가급적 이를 피하도록 코드를 구성해야 한다.

또한 Nvidia GPU의 공유 메모리는 32개의 bank로 구성되어 각각 독립적인 접근이 가능하다. 따라서, 그림 4에 보이는 것처럼 배열의 동일한 열(column)이 서로 다른 bank에 할당되도록 함으로써, 각 쓰레드들이 동시에 같은 열의 데이터에 접근할 때 bank 충돌을 방지하도록 코드를 설계해야 한다.

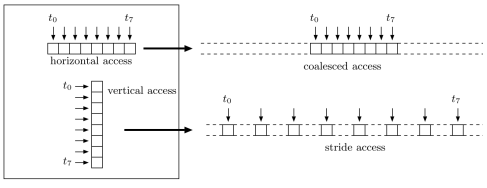

Fig. 3. Coalesced memory access in CUDA

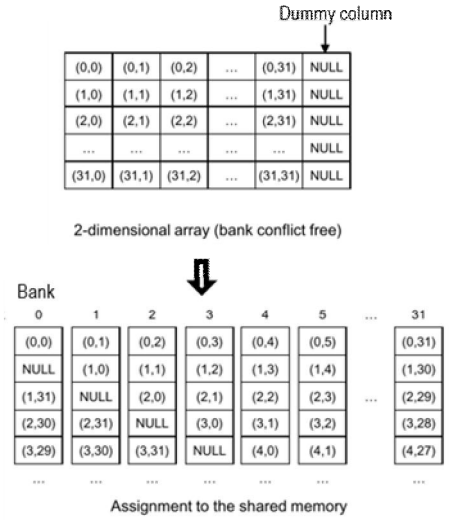

Fig. 4. Bank conflict-free array example

본 논문에서는 Nvidia에서 제공하는 CUDA(compute unified device architecture) 프레임워크를 사용해 병렬 프로그램을 개발했다. CUDA는 다수의 쓰레드들의 집합인 쓰레드 블록(thread block)의 개념을 지원하는데, 한 블럭은 한 개의 SM에 배치되고, 한 개의 쓰레드는 그 안에서 하나의 코어(core)에서 실행된다.

한편 SM 내에서는 각 쓰레드 블럭을 구성하는 쓰레드들을 워프(warp)라고 부르는 32개의 쓰레드 단위로 나누어 이들을 차례로 스케줄링을 한다. 따라서, 각 SM에 충분한 블럭들이 배치되지 않는 경우 자연히 스케줄링 워프의 개수도 적어지게 되어 SM 자원을 효과적으로 활용하지 못하는 문제가 나타나므로, 프로그램 개발자는 GPU의 전체 SM에 고르게 많은 쓰레드 블럭들이 생성되도록 알고리즘을 설계해야 한다.

III. GPU-based Parallel ACS

TSP를 위한 ACS 알고리즘에서 가장 큰 비용은 산술연산이 아닌 메모리 접근 비용이다. 따라서 GPU가 제공하는 메모리 계층 구조를 활용하여 자주 사용되는 전역 메모리 데이터를 공유 메모리에 복사하여 캐싱(caching) 효과를 적극 활용하고, 2장에서 설명한 전역 메모리의 병합 접근과 공유 메모리의 뱅크 충돌(bank conflict)을 줄임으로써 알고리즘의 수행 속도를 높이는 방법이다. 본 연구에서 제안하는 알고리즘은 여정 초기화(tour initialization), 방문 적합도 계산(visit fitness calculation), 여정 구성(tour construction), 여정 정리(wrap-up tour), 페로몬 업데이트(pheromone update)의 5개 커널로 구성되고, 이 커널들을 일정 횟수 동안 반복 실행함으로써 TSP의 최단 경로를 계속 업데이트한다.

본 연구에서 사용한 도시 그래프는 임의의 두 도시 간에 경로가 존재한다고 가정한다. 즉, 어떤 도시 v_i로부터 임의의 다른 도시 v_j로 가는 경로가 존재하며 이 커널은 각 도시 v_i의 위치가 (x_i, y_i)로 표현된 n개의 도시에 대해 임의의 구간 거리 $d_{i,j}$(> 0)를 구하고 해당 구간의 페로몬 $\tau_{i,j}$를 적당한 초기 상수(여기서는 $1/n$)으로 설정한다.

3.1 Tour initialization kernel

각 개미별 초기 도시 위치를 정하고 여정의 경로 방문 정보와 경로 길이를 초기화한다. 본 구현에서는 블록 크기(=블럭 내 쓰레드의 수)를 32개로 설정하고 개미의 수(n_{ant})를 32로 나눈 수, 즉 $\lceil n_{ant}/32 \rceil$ 개의 블록들을 생성하는 CUDA 커널을 작성한다.

3.2 Fitness calculation kernel

임의의 도시 v_i에서 나머지 모든 도시 v_j에 대해 방문 적합도(fitness)를 다음과 같이 계산한다. 이 식에서 α, β는 페로몬과 거리의 영향력을 각각 통제하기 위한 상수이다.

$$fitness = (\tau_{i,j})^{\alpha}(1/d_{i,j})^{\beta}$$

이 커널은 전체 도시 행렬 $n \times n$에 대해 각 32×32 크기의 블록을 $\lceil n/32 \rceil \times \lceil n/32 \rceil$ 개 병렬 생성한다. 이때 생성된 블록의 각 쓰레드는 자신이 담당한 도시 구간 (i,j)에 대해 전역 메모리의 배열에서 해당 페로몬 값 $\tau_{i,j}$와 구간 거리 $d_{i,j}$를 읽어 위의 식을 계산하므로 자연스럽게 메모리 병합 접근이 활용될 수 있다.

3.3 Tour construction kernel

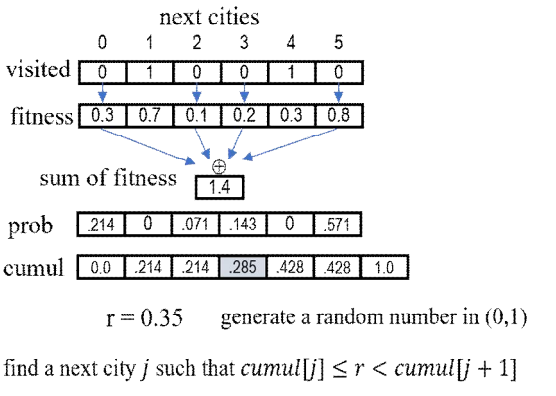

Fig. 5. selection process example for a next visiting city

각 개미 쓰레드는 자신이 탐색할 여정을 확률 과정(stochastic process)을 통해 구성한다. 그림 5는 6개의 인접 도시에서 1개를 선택하는 과정을 예로 보여준다. 우선 visited 배열은 각 도시에 대해 이미 방문했는지 여부를 기록하고, fitness 배열은 3.2절의 과정을 통해 각 도시의 선택 적합도(또는 강도)를 나타낸다.

선택 과정에 앞서 visited, fitness 배열을 통해 각 도시에 대한 선택 확률을 구한다. 이를 위해 이미 방문한 도시는 제외하고, 나머지 도시들의 fitness 값의 합한 후, 이 합계(sum of fitness)에 대한 각 fitness의 비율을 해당 도시의 선택 확률(prob)을 계산한다. 당연히 이미 방문한 도시의 확률은 0이다. 그리고, 다음 계산식을 사용하여 누적확률분포(cumul)를 구한다.

$$cumul[i] = cumul[i-1] + prob[i-1],$$
$$(cumul[-1] = 0, prob[-1] = 0)$$

다음 단계에서 (0, 1) 사이의 난수(random number)를 생성하고, cumul 배열에서 이 값이 속하는 구간을 구해 다음 도시를 결정하게 된다. 이 때 난수 생성 함수는 CUDA가 제공하는 cudaRandom 함수를 사용하는데, 이 함수는 각 개미 쓰레드마다 서로 다른 씨앗값(seed)을 설정하고 이를 curandState 객체를 통해 관리한다.

현재 구현한 알고리즘은 개미별로 여정을 생성하기 때문에 개미의 수가 병렬성을 제한한다. 또한, 각 여정 생성 시 다음 도시를 선택하는 과정이 순차적으로 실행됨으로써 병렬 효과를 기대하기 어려운 문제가 있다. 그럼에도 현재 각 블록을 32개의 쓰레드로 구성하여 전체 개미들을 블록으로 나누어 구성함으로써 GPU의 블록 점유도(block occupancy)를 비교적 높게 유지했다. 이에 반해, 만일 한 여정의 병렬화를 극대화하기 위해 도시의 수만큼의 쓰레드로 구성된 블록을 생성하는 경우, 이 블록이 하나의 SM의 스케쥴 자원을 과도하게 사용하여 오히려 성능이 크게 느려짐을 확인할 수 있었다.

3.4 Tour wrap-up kernel

3.2절의 과정에서 얻어진 각 여정의 길이는 그때까지 발견된 최단 경로(best-path-so-far)와 비교되고 더 짧은 경로일 경우 갱신된다. 이 과정 역시 단일 변수 best-path-so-far에 대한 각 쓰레드의 동시 접근이 이루어지므로, 이의 동기화를 제어하기 위해 CUDA에서 제공하는 atomicMin 함수를 사용한다. 그러나, 이 함수의 사용 시 쓰레드의 병행성(concurrency)이 낮아져 병렬화에 의한 성능 향상을 제약하고 있다.

또한, 이번에 이동한 여정을 구성하는 각 경로에 포함된 구간 (v_i, v_j)에 페로몬의 증가량을 기록하는 2차원 배열 $\delta_{i,j}$에 기록한다. 이 커널 역시 각 블록을 32개의 쓰레드로 구성하고 전체 개미들을 이 크기의 블록들로 나누어 구성하였다.

3.5 Pheromone update kernel

3.4절의 과정을 거쳐 각 개미가 선택한 여정에 대한 평가가 이뤄지면, 전체 경로에 대해 페로몬 영향력을 업데이트해야 한다. 여기에는 크게 (i) 시간이 지남에 따른 페로몬의 감소와 (ii) 앞서 각 개미가 선택한 경로에 더해진 페로몬의 양($\delta[i][j]$)을 다음과 같이 반영한다.

(i) $\tau_{i,j} = (1-\rho)\,\tau_{i,j}$ (이때, ρ는 페로몬의 증발율 상수)

(ii) $\tau_{i,j} = \tau_{i,j} + \delta_{i,j}$

이 커널 역시 전체 도시 행렬 $n \times n$에 대해 각 32×32 크기의 블록을 $\lceil n/32 \rceil \times \lceil n/32 \rceil$ 개 병렬 생성한다. 이 때 생성된 블록의 각 쓰레드는 자신이 담당한 도시 구간 (i,j)에 대해 전역 메모리의 배열에서 해당 페로몬 값 $\tau_{i,j}$와 구간 페로몬 증가량 $\delta_{i,j}$를 읽어 위의 식을 계산하므로 병합 접근이 효과적으로 활용될 수 있다.

IV. Experiments and Analysis

본 절에서는 GPU 기반 병렬 알고리즘의 성능을 살펴보기 위해 우선 본래 ACS 알고리즘의 CPU 버전에 대한 성능을 살펴보고, 개미의 수에 따른 실행 시간과 구해진 해답의 품질을 살펴본다. 그리고, GPU 실행 시간에서 각 부분이 차지하는 실행 시간의 비율을 분석한다.

CPU에서 실행될 순차 ACS 알고리즘은 C 언어로 구현된 공개 코드를 참고하여 구현하였으며[15], GPU 병렬 알고리즘은 Nvidia에서 제공하는 CUDA library(version 7.5)를 사용하여 Windows Visual Studio 2019 환경에서 개발하였고, 표 1에 보인 하드웨어 시스템에서 실험하였다.

ACS 알고리즘의 동작을 통제하는 파라미터는 관련 연구에서 제시한 값들을 참조하여 다음과 같이 설정하였다. 페로몬과 거리에 따른 선택 적합도를 제어하는 α, β는 각각 0.2와 3.0, 페로몬의 증발율 상수 ρ는 0.01로 설정하였다.

또한 실험에 사용된 TSP 맵은 TSPLIB[16]으로부터 bier127, lin318, ali535, pr1002 등 4개 데이터를 선택하였고(각 데이터 이름에 표시된 숫자는 도시의 수), 개미의 수는 도시의 수와 같은 수로 설정하였다.

Table 1. System Specification for Experiments

	CPU 시스템	GPU 시스템
Processor	Intel Core i9-9900K	Nvidia Titan RTX (Turing Architecture)
Clock	3.6 GHz	1.750 GHz
# Cores	8	4608 (72 SMs x 64 cores/SM)
L1 cache	64 KB (per core)	64 KB (per SM)
L2 cache	256 KB (per core)	6 MB
Memory bandwidth	41.6 GB/s	672 GB/s
Memory capacity	64 GB (on-board)	24 GB (on-device)
Performance	49.45 GFLOPS	16312.32 GFLOPS

4.1 Parallelization performance

ACS를 이용한 TSP 알고리즘은 여러 차례의 반복 실행을 통해 근사 최적해를 구하지만, 알고리즘의 성능 비교를 위해서는 1회 반복에 소요된 실행 시간을 측정하였다.

표 2에 보이는 바와 같이, bier127을 제외하고 대체로 10.13~11.37배의 성능 향상을 보였는데, 이는 도시 간 거리 정보와 페로몬 잔량을 저장하는 $N_{city} \times N_{city}$ 행렬에 대해 32x32 병렬 블록에 의한 데이터 분산처리 효과와 연속 메모리 공간에 대한 병합 접근 효과로 분석된다.

다만, bier127의 경우 4.82배의 비교적 낮은 성능향상을 보인 것은 앞서 언급한 병렬처리 효과에 비해 여정 생성 과정에서 다음 방문 도시를 선정하는 순차적 과정이 차지하는 비중이 상대적으로 높고, 페로몬 업데이트 과정에서 사용한 atomicMin 함수가 상당한 비용을 요구하기 때문으로 분석된다.

Table 2. Parallelization Performance [ms]

data	CPU	GPU	speedup
bier127	9.02	1.87	4.82
lin318	23.84	2.35	10.13
ali535	42.72	3.91	10.92
pr1002	209.12	18.39	11.37

4.2 Performance analysis

현재 실험에서는 반복 횟수를 30회로 한정하고, 개미의 수는 도시의 수와 동일하게 생성한 상태에서 각 횟수마다 계산된 최단경로(best-path-so-far)를 살펴보았다. data에 따라 약간의 차이는 있었지만 그림 6에 보이는 바와 같이 대개 10회 내외에서 결과가 수렴하는 것으로 나타났고, bier127의 경우에는 1회의 결과가 최종 결과로 연결되었다. 또한 크기가 큰 데이터들의 경우에는 처음 5회 반복 근처에서 최단 경로로 급격히 수렴하는 현상을 보였다.

개미의 수, 즉 병렬처리 쓰레드의 수에 따른 반복실행

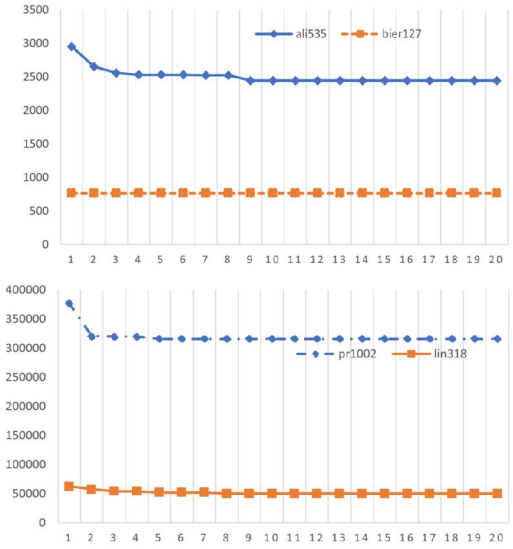

Fig. 6. Computation results with iterations

결과와 그 품질을 좀더 살펴보기 위해 적당한 데이터 크기를 갖는 ali535 데이터를 대상으로 개미의 수를 260, 535, 1050개로 각각 변화시키며 실험을 수행하였다.

그림 7의 결과를 보면 도시의 수보다 많은 535개 이상의 개미를 활용하는 것은 최단경로의 품질을 높이는데 크게 기여하지 않는 것으로 보인다. 다만 도시 수의 2배인 1070개의 개미를 활용한 경우 상당히 일찍 최적값을 향해 수렴하는 경향을 보였다. 그러나 최종 결과값은 2423으로 535개를 사용했을 때의 결과 2438과 큰 차이를 보이지 않았다.

이에 반해 도시의 수의 약 절반에 해당하는 260개의 개미를 사용한 경우에 초반 결과는 비교적 큰 값인 4042로 시작하고 3230, 3125, 2964, 2931, ... 등으로 점진적으로 수렴하는 모습을 보이고 있다. 이것은 낮은 수준의 병렬처리 성능이 반복 횟수에 의해 일정 부분 보상받음을 보여준다. 그럼에도 최종 결과는 앞서의 535, 1070개의 실행 결과보다 좋지 않은 2558로 끝나, 개미의 수가 도시의 수보다 크게 적을 경우 결과값의 품질에 한계가 있고, 이것은 단순히 반복 횟수를 증가시킴으로써 해결될 수 없음을 보여준다.

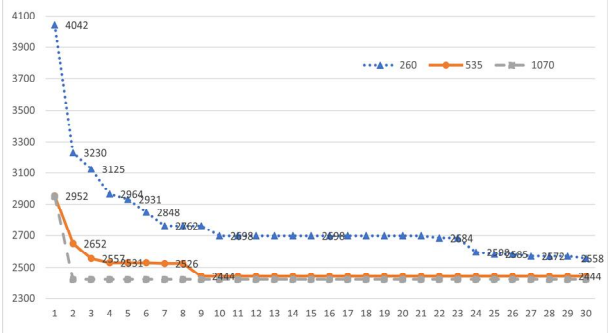

Fig. 7. Results trend with the number of ants

위의 결과를 볼 때, 일정한 품질(최적값 대비 일정 오차 이내)의 결과를 얻기 위해 ACS 알고리즘은 적어도 데이터 크기에 상응하는 개미와 일정 횟수 이상의 반복 계산이 절대적임을 확인할 수 있다. 특히 개미의 수(병렬처리 성능)가 일정 수준 이상으로 제공되어야 고품질의 결과를 보장받을 수 있다는 점은 CPU 기반으로 ACS를 구현하는 것이 성능면에서 한계에 다다랐음을 보여준다. 더욱이 수백 또는 수천 개의 데이터에 대해 ACS 알고리즘을 수행할 때, CPU에서 개미의 수를 그 수준으로 생성, 활용한다는 것은 그 수 만큼의 반복 연산을 일으켜 결국 GPU 기반 병렬 알고리즘과 성능 차이는 더욱 크게 벌어질 것으로 예상된다.

V. Conclusions

ACS는 메타 휴리스틱을 사용해 TSP와 같이 계산 복잡도가 높은 문제의 근사 최적해(near optimal solution)를 널리 활용되며, 특히 지속적으로 네트워크의 상태가 변화하는 동적인 TSP 환경에 ACS가 매우 효과적이어서 다양한 응용분야의 적용이 기대된다. 그럼에도, 여전히 TSP 문제 자체는 물론 ACS가 갖는 막대한 계산량으로 인해 이의 효과적인 병렬화는 필수적이다. 본 논문에서는 GPU가 제공하는 데이터 병렬성을 최대한 활용한 알고리즘을 설계하고, 이를 Nvidia GPU 상에서 구현하고 실험하였다.

공개된 주요 TSP 데이터를 대상으로 실험한 결과, 단일 CPU에서 동작하는 알고리즘에 비해 GPU 병렬 알고리즘은 약 10.13~11.37배의 성능 개선 효과를 보였다. 다만, bier127와 같이 비교적 작은 규모의 데이터에 대해 4.82배의 비교적 낮은 성능향상을 보인 것은 여정 생성 과정에서 다음 방문 도시를 선정하는 순차적 과정이 차지하는 비중이 상대적으로 높았던 것으로 보인다.

실험을 통해 일정한 품질(최적값 대비 일정 오차 이내)의 결과를 얻기 위해 ACS 알고리즘은 적어도 데이터 크기에 상응하는 개미와 일정 횟수 이상의 반복 계산이 절대적임을 확인했다. 특히 개미의 수(병렬처리 성능)가 일정 수준 이상으로 제공되어야 고품질의 결과를 보장받을 수 있다는 점은 CPU 기반으로 ACS를 구현하는 것이 성능면에서 한계에 다다랐음을 보여준다. 더욱이 수백 또는 수천 개의 데이터에 대해 ACS 알고리즘을 수행할 때, CPU에서 개미의 수를 그 수준으로 생성, 활용한다는 것은 그 수 만큼의 반복 연산을 일으켜 결국 GPU 기반 병렬 알고리즘과 성능 차이는 더욱 크게 벌어질 것으로 예상된다.

특히 대규모 TSP 맵 데이터에서 다음 방문 도시의 확률적 선택 적합도를 계산하거나 개미들의 움직임에 따른 페로몬의 자취를 업데이트하는 과정은 GPU의 데이터 병렬구조에 아주 적합한 계산과정으로 성능 개선에 주요한 부분이었다. 다만, 개미들이 매번 다음 여정을 생성하는 과정은 순차적 확률 과정의 연속으로 병렬화에 한계가 있었다. 따라서, 향후 연구에서는 이 부분의 알고리즘 속성을 세분화하고 더 적극적으로 병렬화 방법을 적용할 필요가 있다.

ACKNOWLEDGEMENT

This work was supported by Hankuk University of Foreign Studies Research Fund of 2020.

REFERENCES

[1] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, “Introduction to Algorithms”, 2nd ed., The MIT Press, 2001.

[2] M. Dorigo, V. Maniezzo, and A. Colorni, “The ant system: Optimization by a colony of cooperating agents”, IEEE Transactions on Systems, Man, and Cybernetics-Part B 26, pp. 29-41, Feb. 1996.

[3] Junqi Yu, Ruolin Li, Zengxi Feng, Anjun Zhao, Zirui Yu, Ziyan Ye, Junfeng Wang, “A Novel Parallel Ant Colony Optimization Algorithm for Warehouse Path Planning”, Journal of Control Science and Engineering, Hindawi, pp. 1687-5249, vol. 2020, Aug. 2020.

[4] J. Kanda, A. D. Carvalho, E. Hruschka, “Meta-learning to select the best meta-heuristic for the Traveling Salesman Problem: A comparison of meta-features”, Neurocomputing, Vol. 205(C), pp. 393-406. Sep. 2016.

[5] X. Chen, Y. Zhou, Z. Tang, “A hybrid algorithm combining glowworm swarm optimization and complete 2-opt algorithm for spherical traveling salesman problems”, Applied Soft Computing, Vol. 58, pp. 104-114, Apr. 2017.

[6] C. Groba, A. Sartal, “Solving the dynamic traveling salesman problem using a genetic algorithm with trajectory prediction: an application to fish aggregating devices,” Computers & Operations Research, pp. 22-32, Apr. 2015.

[7] Mavrovouniotis, M., Yang, S., Van, M., Li, C., Polycarpou, M., “Ant Colony Optimization Algorithms for Dynamic Optimization: A Case Study of the Dynamic Travelling Salesperson Problem”, IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE, 15(1), 52-63, Feb. 2020.

[8] M. Mavrovouniotis, F. M. Muller, and S. Yang, “Ant colony optimization with local search for dynamic traveling salesman

problems," IEEE Trans. Cybern., vol. 47, no. 7, pp. 1743-1756, July 2017.

[9] M. Manfrin, M. Birattari, T. St¨utzle, and M. Dorigo, "Parallel Ant Colony Optimization for the Traveling Salesman Problem", in Proc. of5th International Workshop on Ant Colony Optimization and Swarm Intelligence, Vol. LNCS 4150, Springer-Verlag, pp. 224-234, Sep. 2006.

[10] P. Delisle, M. Krahecki, M. Gravel, and C. Gagn´e, "Parallel implementation of an ant colony optimization metaheuristic with OpenMP", in Proc. of the 3rd European Workshop on OpenMP, pp. 1-7, Jan. 2001.

[11] Breno A. de Melo Menezes, Nina Herrmann, Herbert Kuchen & Fernando Buarque de Lima Neto, "High-Level Parallel Ant Colony Optimization with Algorithmic Skeletons", International Journal of Parallel Programming, vol. 49, pp. 776-801, Dec. 2021.

[12] A. Del´evacq, P. Delisle, M. Gravel, and M. Krahecki, "Parallel Ant Colony Optimization on Graphics Processing Units", Journal of Parallel and Distributed Computing, vol. 73, no. 1, pp 52-61, Jan. 2013.

[13] Menezes, B.A., Kuchen, H., Neto, and de Lima Neto, "Parallelization strategies for GPU-based ant colony optimization solving the traveling salesman problem", In Proc. of IEEE Congress on Evolutionary Computation, pp. 3094-3101, June 2019.

[14] Yi Zhou, Fazhi He, Neng Hou, Yimin Qiu, "Parallel ant colony optimization on multi-core SIMD CPUs", Future Generation Computer Systems, vol. 79, part. 2, pp. 473-487, Feb. 2018,

[15] Stützle Thomas. Ant colony optimization - http://iridia.ulb.ac.be /~mdorigo/ACO/aco-code/ public-software.html. [accessed: Dec. 05, 2021]

[16] TSPLIB, "Symmetric traveling salesman problem", http://elib.z ib.de/pub/mp-testdata/tsp/tsplib/ tsplib.html. [accessed: Nov. 22, 2021]

Authors

Yunseok Rhee received the B.S. degree in Computer Science and Statistics from Seoul National University, Korea in 1984 and the Ph.D degree in Computer Science from KAIST, Korea in 1995 and 1999 respectively. He is currently a Professor in the Division of Computer Engineering at Hankuk University of Foreign Studies. His current research interests include parallel and distributed systems, operating systems and internet services.