CUDA 기반 공유 메모리를 활용한 병렬 숄레스키 분해 구현

강준범[1], 이명호[2], 박능수[1]
[1]건국대학교 컴퓨터공학과
[2]명지대학교 컴퓨터공학과
aopko@konkuk.ac.kr, myunghol@mju.ac.kr, neungsoo@konkuk.ac.kr

CUDA-Based Shared-Memory Implementation of Parallel Cholesky Decomposition

Junbeom Kang[1], Myungho Lee[2], Neungsoo Park[1]
[1]Dept. of Computer Science and Engineering, Konkuk University
[2]Dept. of Computer Science and Engineering, Myongji University

요 약

본 논문에서는 CUDA 기반의 공유 메모리를 활용한 병렬 숄레스키(Cholesky) 분해 알고리즘을 제안한다. 기존 CUDA 라이브러리 기반 접근법(cuSolver, MAGMA 등)이 공유 메모리 활용이 제한적이고 Warp Divergence 및 비효율적 메모리 접근 문제를 가질 수 있다는 점을 개선하고자, 본 연구에서는 N-블록 공유 메모리 활용 알고리즘을 설계하였다. 제안된 알고리즘은 POTRF-TRSM 단계에서 공유 메모리를 적극 활용해 전역 메모리 접근을 최소화하고, 명시적인 트레일링 연산 없이도 종속 문제를 완화하여 병렬성을 높인다. RTX 4070Ti Super 환경에서 Right-looking 및 Left-looking 방식과 비교 실험을 수행한 결과, nb=32의 경우 최대 4배 이상의 성능 향상이 관찰되었으며, 행렬 크기와 nb가 증가할수록 병렬화 효과가 더욱 두드러짐을 확인하였다.

1. 서론

행렬 분해(Matrix Factorization)는 과학·공학·금융·머신러닝 등 다양한 분야에서 필수적인 연산이다. 이 중 대칭 양의 정부호(Symmetric Positive Definite, SPD) 행렬을 효율적으로 분해하는 숄레스키 분해(Cholesky Decomposition)는 수치해석, 최적화 문제, 통계적 모델링 등에 활용되고 있다. 그러나 행렬의 규모가 커짐에 따라 CPU 기반의 연산 모델의 한계가 도래하게 되었으며, 병목 현상이 발생하는 문제가 초래되었다. 이 한계를 극복하고자 GPGPU를 이용한 병렬 연산이 활발히 연구되고 있으며, NVIDIA사의 CUDA(Computed Unified Device Architecture)는 이를 구현할 수 있는 대표적인 GPGPU 프로그래밍 모델로 자리 잡았다.

기존 CUDA 기반 숄레스키 분해 연구들은 cuSolver나 MAGMA와 같은 CUDA 자체적인 라이브러리 기반의 연구로 진행되었다. 그러나 이러한 연구들에서는 공유 메모리 활용이 제한적이거나, 블록 위치에 따른 병렬화가 균형적이지 않아 비효율적인 메모리 접근이나 Warp Divergence 문제가 발생하는 가능성이 존재하게 되었다.

따라서 본 연구에서는 이러한 한계를 해결하기 위해 공유 메모리를 활용한 N-블록 기반 숄레스키 분해 알고리즘을 제안한다. 제안한 알고리즘은 선제적으로 숄레스키 분해 연산을 작은 창에 먼저 수행시켜 공유메모리에 적재해 이 연산 결과에 종속적인 원소들이 숄레스키 분해 연산 수행을 위해 접근 시 글로벌 메모리 접근을 최소화하는 이점을 극대화 하는 알고리즘이다. 이를 통해 연산 효율성을 높이고 메모리 대역폭 병목을 완화한다.

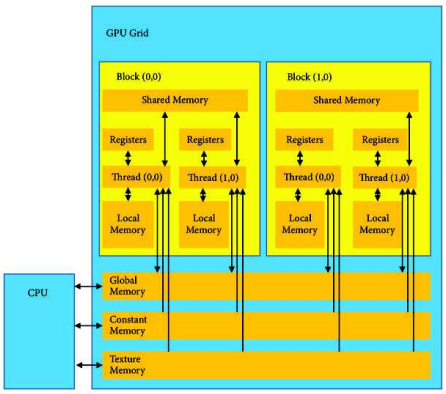
2. 배경지식

2.1 CUDA GPU Memory Architecture

서론에서 설명한 CUDA 에서는 GPGPU의 다양한 메모리에 접근할 수 있도록 설계할 수 있다.

전역 메모리는 장치 전체가 접근 가능한 대용량 DRAM이다. 대역폭은 높지만 지연 또한 크다. 워프 단위의 Coalescing(연속 주소 접근에 대한 단일 트랜잭션 묶음)이 필수적이다. 공유 메모리는 SM(Streaming Multiprocessor) 온칩에 위치한 블록 수준의 고속 스크래치패드로 지연이 매우 낮고 명시

적으로 관리가 가능하여 타일링 기반 알고리즘(블록 행렬 연산)에 적합하다. 32개의 뱅크(bank)로 구성되며 같은 사이클에 동일 뱅크를 여러 스레드가 다른 주소로 접근하면 뱅크 충돌(bank-conflict)로 직렬화가 발생할 수 있다. 레지스터는 스레드 전용의 가장 빠른 저장소이며, 레지스터가 초과 사용되는 경우 전역 메모리 가상 공간에 위치한 로컬 메모리가 사용된다. L1, L2 캐시 메모리는 각각 코어 인접 구역과 칩 전역으로 캐시 메모리를 제공한다. L1 캐시 메모리는 공유 메모리와 자원을 공유하는 특징을 가진다. 상수/텍스쳐/읽기 전용 메모리의 경우는 각각 커널 실행 간 상수를 저장, 공간적 지역성(2D/3D)에 최적화되어 이미지 데이터 인덱싱, 큰 배열에 읽기만 수행하는 메모리이다.[1]



(그림 1) CUDA GPU Memory Architecture[2]

3. 알고리즘 제안

3.1 숄레스키 분해

```
for k = 0 .. N-1:
    s = 0
    for j = 0 .. k-1:
        s += A[k,j] * A[k,j]
    A[k,k] = sqrt( A[k,k] - s ) ... (1)

    for i = k+1 .. N-1:
        s = 0
        for j = 0 .. k-1:
            s += A[i,j] * A[k,j]
        A[i,k] = ( A[i,k] - s ) / A[k,k] ... (2)
```

(코드 1) Pseudo-code

코드 1은 일반적인 직렬 숄레스키 분해를 Pseudo-code 형식으로 변환하여 작성된 것이다. 특정 원소값을 구하기 위해선 동일한 행에 위치한 앞 열 데이터가 필요함을 코드 1의 (1), (2)를 통해 알 수 있다. 이런 데이터 종속성에 의해 숄레스키 분해를 병렬화 하기에는 까다로운 조건이 있다.

3.2 공유 메모리를 활용한 숄레스키 분해 알고리즘

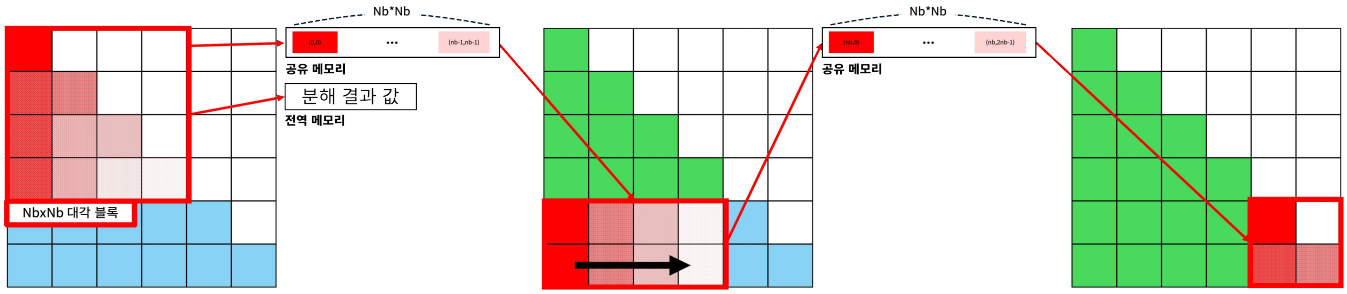
```
procedure PARALLEL_BLOCK_CHOLESKY(A, N, nb):
    for k = 0 to N-1 step nb:
        kb = min(nb, N - k)
        Akk = A[k:k+kb, k:k+kb]
        for t = 0 to k-1 step nb:
            tb = min(nb, N - t)
            Lkt = A[k:k+kb, t:t+tb]
            Akk = Akk - (Lkt * Lkt^T)

        Lkk = POTRF(Akk)
        A[k:k+kb, k:k+kb] = Lkk

        for i = k + kb to N-1 step nb:
            ib = min(nb, N - i)
            Aik = A[i:i+ib, k:k+kb]
            for t = 0 to k-1 step nb:
                tb = min(nb, N - t)
                Lit = A[i:i+ib, t:t+tb]
                Lkt = A[k:k+kb, t:t+tb]
                Aik = Aik - (Lit * Lkt^T)
            Lik = TRSM_TRANSPOSE(Lkk, Aik)
            A[i:i+ib, k:k+kb] = Lik
    end for
end procedure
```

(코드 2) Parallel Cholesky Pseudo-code

3.1에서 설명한 다양한 이유를 근거로 병렬화를 위해 고려해야 하는 많은 조건 중 위 알고리즘은 패널 방식과 공유 메모리 활용을 통해 가속화를 유도했다. 일반적인 Right-looking Cholesky 방식[3]에 더불어 우수한 병렬 컴퓨팅 파워를 보유하고 있는 GPGPU와 최근 상당히 상승한 GPU 내부의 공유 메모리 크기를 적극적으로 활용하고 있다. 프로그램의 흐름은 다음과 같다. 먼저 사용자가 지정한 nb 변수의 크기만큼 창(Window)을 만들어 숄레스키 연산(POTRF)을 수행한다. 이후 해당 창은 공유 메모리에 저장되고, 다음 단계에서 이 공유 메모리의 값을 그대로 활용하여 하단 열의 값을 구하는 연산(TRSM)에 사용한다. TRSM 단계에서 계산된 결과값은 다음 POTRF 연산을 위해 nb*nb 크기만큼 공유메모리에 저장되어 명시적인 트레일링이 필요없도록 하고, 계산을 위해 이전 열에 접근할 때 필수적으로 발생하는 전역 메모리 접근을 최소화한다. 실질적으로는, 그림 2 에서의 첫 번째 단계와 두 번째 단계가 지속적으로 반복되어 발생하며 숄레스키 분해를 진행하게 되는 것이다. 더 이상 TRSM 연산이 필요 없게 되면 모든 결과값을 전역 메모리에서 CPU 메모리로 전달한다.



(그림 2) Parallel Cholesky Decomposition

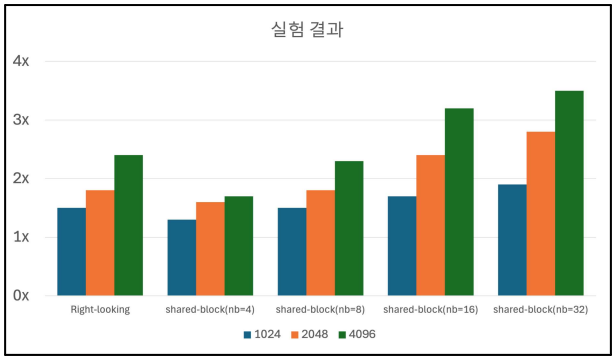
3.3 실험환경

표 1은 실험을 진행한 하드웨어 사양과 소프트웨어 환경을 나타내고, 표 2는 GPGPU의 주요 세부 성능 지표를 나타내고 있다.

<표 1> Experimental Environment

Hardware Specification	
CPU	AMD Ryzen 7 7800x3D
GPU	NVIDIA RTX 4070Ti Super
RAM	DDR5 32GB
Software Environment	
OS	Windows 11 / WSL(Ubuntu)
CUDA	8.9 version
CUDA Toolkit	12.6 version
측정 프로그램	CUDA Nsight System

3.4 실험 결과



(그래프 1) 실험 결과

위 논문에서 제시하는 알고리즘의 성능 평가를 위해서 비교 대상으로 타일링 크기만을 고려하는 단순한 구조의 Right-looking/Left-looking(RL/LL) 방식을 구현했으며, 파라미터값과 원행렬의 크기를 조절하며 성능을 비교했다. 원행렬(NxN)의 한 변의 크기는 범위를 {1024,2048,4096} 창의 한 변의 크기는 {4,8,16,32} 범위 내로 설정했고, 원소의 자료형으로는 FP32로 설정했다. 프로그램 실행 시간은 NVIDIA에서 제공하는 Nsight System을 활용했으며 이를 통해 세밀한 커널의 실행 시간을 계산했다. 차트의 세로 축 수치는 LL 코드의 실행 시간을 기준으로 비교했을 때 보이는 가속 비율을 의미한다. 그래프 1을 통해 실험 결과를 분석해보면, 창의 크기가 작은 경우에는 성능 향상이 잘 발생하지 않는데, 이 경우는 공유 메모리의 접근 오버헤드가 다수 발생했음을 추론할 수 있다. 그러나, 배열의 크기가 증가하고, 창의 크기가 커질수록 가속화가 크게 발생하며 성능 향상이 유의미하게 발생함을 보이고 있다.

4. 결론

본 연구에서는 GPU 공유 메모리를 활용한 N-블록 기반 병렬 숄레스키 분해 알고리즘을 제안하고, 전역 메모리 상 병렬화 방식과 비교하여 가속화 수준을 평가했다. 실험 결과를 통해서 공유 메모리 N-블록 병렬 숄레스키 분해는 실행 시간을 유의미하게 단축시켰음을 확인할 수 있었다. 특히 행렬의 크기가 커지고, 창의 크기가 커질수록 성능 향상이 더욱 뚜렷하게 나타났다. 추후 연구에서는 Multi-GPU 환경에서의 확장성을 검증함으로써 보다 범용적이고 고성능의 병렬 행렬 분해 알고리즘으로 발전시킬 수 있을 것이다.

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 정보통신방송혁신인재양성(메타버스융합대학원)사업 연구 결과로 수행되었음
(IITP-2025-RS-2023-00256615)

참고문헌

[1] NVIDIA “CUDA C++ Programming Guide” v.13.0, Chapter 20 Compute Capabilities https://developer.nvidia.com/cuda-toolkit-archive

[2] Jianqi Lai et al. “A Multi-GPU Parallel Algorithm in Hypersonic Flow Computations” Mathematical Problems in Engineering Volume 2019. pp.15

[3] Edward Anderson, Jack Dongarra, “Evaluating Block Algorithm Variants in LAPACK”, SIAM, Chicago, 1989, pp.3-8