Implementation Results

202011035 김민수

img_128.bmp	img_256.bmp	img_512.bmp	img_768.bmp	img_1024.bmp
Speedup: 1.731189	Speedup: 1.733317	Speedup: 1.796345	Speedup: 1.807390	Speedup: 1.910548
Speedup: 1.706828	Speedup: 1.729118	Speedup: 1.755197	Speedup: 1.867218	Speedup: 1.905607
Speedup: 1.691504	Speedup: 1.730294	Speedup: 1.775499	Speedup: 1.817863	Speedup: 1.879732
Speedup: 1.675914	Speedup: 1.758604	Speedup: 1.780204	Speedup: 1.816245	Speedup: 1.919969
Speedup: 1.712796	Speedup: 1.759349	Speedup: 1.796136	Speedup: 1.816796	Speedup: 1.995160
평균: 1.703646	평균: 1.742136	평균: 1.780676	평균: 1.825102	평균: 1.922203

1. 루프 언롤링의 상세 구현

루프 언롤링은 반복문의 작업을 줄여 루프 오버헤드를 감소시키는 기법입니다. 특히, 이미지 처리에서는 픽셀 단위의 반복 작업이 많기 때문에, 한 번의 루프에서 다수의 픽셀을 처리하면 루프의 시작과 끝에서 발생하는 오버헤드를 줄일 수 있습니다.

루프 언롤링의 추가적인 효과는 CPU 파이프라인의 효율성을 높이는 데 있습니다. CPU는 명령어를 파이프라인에 넣고 처리하는 과정에서 분기 예측을 수행합니다. 루프 언롤링을 통해 분기의 빈도를 낮추면 파이프라인의 효율성이 증가합니다. 이는 이미지 처리 작업에서 특히 유리하며, 각 픽셀을 처리하는 시간이 단축됩니다.

2. 메모리 접근 최적화의 상세 구현

메모리 접근 패턴을 최적화하는 것은 캐시 히트율을 높이는 데 중요합니다. 캐시는 메모리의 데이터 블록을 로드하여 처리 속도를 높이기 때문에, 연속적인 메모리 접근 패턴을 가지면 캐시 히트율이 증가합니다.

특히 행 우선 접근 방식은 2차원 배열을 1차원 배열처럼 연속적으로 접근하도록 하여 캐시의 효율을 극대화합니다. 이 방법은 큰 이미지 파일을 처리할 때 특히 효과적이며, 캐시 미스를 줄여 성능을 향상시킵니다. 예를 들어, 2차원 배열의 각 행을 순차적으로 접근하면, 한 번의 캐시 로드로 여러 데이터를 사용할 수 있습니다.

3. 병렬 처리의 상세 구현

멀티스레딩을 통한 병렬 처리는 대용량 데이터 처리에서 매우 효과적입니다. OpenMP를 사용하면 코드의 병렬화를 비교적 간단하게 구현할 수 있습니다. OpenMP는 코드의 병렬화 영역을 지정하고, 병렬 처리를 관리하는 데 필요한 대부분의 작업을 자동으로 수행합니다.

병렬 처리는 각 스레드가 독립적으로 작업을 수행할 수 있을 때 가장 효과적입니다. 이미지 처리의 경우, 각 픽셀을 독립적으로 처리할 수 있기 때문에 병렬화가 매우 용이합니다. 이를 통해 전체 처리 시간을 단축시킬 수 있으며, 특히 고해상도 이미지나 대량의 이미지 파일을 처리할 때 유용합니다.

4. 결론

이번 최적화 작업을 통해 기준 대비 최대 1.92배의 속도 향상을 달성할 수 있었습니다. 다양한 이미지 크기에 대해 일관된 성능 향상을 이루었으며, 이는 다음과 같은 세 가지 주요 전략 덕분입니다.

루프 언롤링(Loop Unrolling): 반복문의 오버헤드를 줄여 루프의 효율성을 극대화했습니다. 이를 통해 CPU의 파이프라인 효율을 높이고 분기 예측 실패를 줄임으로써 성능을 향상시켰습니다.

메모리 접근 최적화(Memory Access Optimization): 데이터의 공간적 지역성을 최적화하여 캐시 히트율을 높였습니다. 행 우선 접근 방식을 사용하여 메모리 접근 패턴을 개선함으로써 캐시 미스를 줄이고 전반적인 처리 속도를 높였습니다.

병렬 처리(Parallel Processing): 멀티스레딩을 도입하여 CPU의 멀티코어 구조를 효율적으로 활용했습니다. OpenMP를 사용하여 이미지 처리 작업을 병렬화함으로써 처리 시간을 크게 단축할 수 있었습니다.

이 세 가지 전략을 결합함으로써 높은 성능 향상을 이끌어낼 수 있었습니다. 또한 사진의 크기가 커질수록 효율이 증가하는 모습을 관측할 수 있었습니다. 이는 루프 언롤링, 메모리 접근 최적화, 병렬 처리와 같은 최적화 전략들이 큰 데이터 세트에서 더 큰 효과를 발휘하기 때문입니다. 작은 이미지에서는 이러한 최적화의 효과가 제한적일 수 있지만, 이미지 크기가 커질수록 반복문 오버헤드, 캐시 미스, 스레드 분할 등의 문제들이 두드러지게 나타나며, 이를 해결하는 최적화 전략들의 효과가 극대화됩니다. 이러한 이유로 큰 이미지에서 더 높은 효율을 달성할 수 있게 됩니다.