creative commons
COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학석사 학위논문

통신 가능한 다중 코어 기반의 병렬화된 영역 성장 기법

Parallel Seeded Region Growing based on Intercommunicating Multi-Core

2013년 2월

서울대학교 대학원
전기·컴퓨터공학부
김 도 현

통신 가능한 다중 코어 기반의 병렬화된 영역 성장 기법

Parallel Seeded Region Growing based on Intercommunicating Multi-Core

지도교수 신 영 길

이 논문을 공학 석사학위논문으로 제출함

2012년 12월

서울대학교 대학원

전기·컴퓨터공학부

김 도 현

김 도 현의 석사학위논문을 인준함

2013년 1월

위 원 장 장 병 탁 (인)

부 위 원 장 신 영 길 (인)

위 원 서 진 욱 (인)

국문 초록

관심 영역을 분할하는 것은 영상을 분석하기 위해 하는 선행 작업으로 많이 쓰이는 방법이다. 그러나 의료 영상 기기들의 정밀도가 높아져 영상 데이터의 크기가 커지면서 분할에 적지 않은 시간이 걸린다. 이를 병렬적으로 처리할 수 있다면 시간이 단축되어 이러한 문제점을 해결할 수 있다.

본 논문에서는 영상 분할 기법 중 하나인 영역 성장 기법(Region Growing)을 병렬적으로 처리하는 방법을 제안한다. 영역 성장 기법은 문제 영역의 크기가 정해져 있지 않고, 시간에 따라 변하기 때문에 일반적인 데이터 병렬 처리 방법을 적용하는 것이 용이하지 않다. 본 논문에서는 이를 코어 간 통신이 가능한 다중 SRP 서브 시스템의 특성을 이용하여 가속하는 기법을 제안한다. 그리고 초기에 코어마다 작업을 균등하고, 빠르게 분배하기 위해 순환 순서 방식(Round Robin)이 아닌 새로운 작업 전파 전략을 사용한다. 실험 결과, 단일 코어 환경과 비교했을 때 약 10.98배의 속도 향상이 있었다.

주요어 : 영상 분할 기법, 영역 성장 기법, 병렬 처리, 다중 SRP 서브 시스템

학번 : 2011-20801

목 차

그림 목차

표 목차

코드 목차

1. 서론

영상을 분석할 때 하는 선행 작업 중 하나가 관심 영역을 분할하는 것이다. 관심 영역 분할하는 작업은 영상을 분석할 때 드는 비용을 줄이기 위해 꼭 필요한 작업이고, 실제로 많은 연구가 이루어지고 있다. 하지만 최근 의료 영상 기기들의 정밀도가 높아져 영상 데이터의 크기가 커지면서 이를 처리하는 시간이 적지 않게 들고 있다. 따라서 이 작업을 병렬적으로 처리하는 것은 처리 비용을 절감하는 측면에서 효과적인 방법이라고 할 수 있다. GPU를 이용한 볼륨 분할 방법 [1] 이 있으나, 관심 영역이 아닌 영역에 대한 연산도 계속 발생되어 효율성은 떨어진다.

영역 성장 기법(Region Growing) [3] 은 씨드 영역 성장 기법(Seeded Region Growing)이라고도 하는데, 이는 영역 성장이 씨앗(seed) 픽셀로부터 성장해가기 때문이다. 이 알고리즘은 영상 분할 기법 중 하나로 적용이 간단하여 널리 쓰이는 기법 중 하나이다. 또한 이는 의료 영상에서 자궁경부암 [4], 대뇌혈관 [5], 유방암 [6] 등을 검출하는데 유용하게 사용된다. 영역 성장 기법에는 여러 가지 특성 값을 이용한 방법들이 있지만 효과적인 분할을 위해 본 논문에서는 텍스처 특성값(texture feature value)을 이용한 방식 [2] 을 사용하였다.

그리고 영역 성장 기법은 씨앗 픽셀이 여러 개가 선택되기도 하고, 자동으로 선택되기도 하는데, 일반적으로 의료 영상에서의 경우 의사가 원하는 관심 영역은 한정적이고, 같은 영상이라고 하더라도 의사에 따라 원하는 관심영역이 다르기 때문에 본 논문에서는 원하는 관심 영역에 씨앗 픽셀을 직접 지정하는 방식을 채택하였다.

최종적으로 병렬화된 영역 성장 기법을 개발하기 위해 다중 코어 중 SRP(Samsung Reconfigurable Processor) [10] 를 단위 코어로 하는 다중

SRP 서브 시스템을 기반으로 두었다.

이는 영역 성장 기법이 일반적인 영상 처리와 달리 문제의 크기가 정해져 있지 않고 과정에 따라 변하기 때문에 일반적인 병렬 처리 방법으로는 적용이 되지 않는데, 다중 SRP 서브시스템의 경우 코어 간의 통신이 가능하다는 특성이 이를 해결하는데 적합하다고 생각을 하여 이를 선택하게 되었다.

이 후의 논문 구성은 다음과 같다. 2장에서는 기존 영역 성장 기법이 어떤 것들이 있는지 알아보고 3장에서는 병렬화된 영역 성장 기법을 다중 SRP 서브 시스템을 이용하여 구현한 세부 사항에 대하여 설명한다. 4장에서는 실험결과를 제시하고 성능을 단일 코어와 비교하여 분석한다. 마지막으로, 5장에서는 실험 결과를 바탕으로 정리를 한 다음 결론을 맺는다.

2. 관련 연구

이 장에서는 영역 성장 기법과 영역 포함 결정 방식, 그리고 개발 기반이 되는 다중 SRP 서브 시스템에 대해서 설명한다.

2.1. 영역 성장 기법 (Region Growing)

영역 성장 기법은 처음에 사용자의 입력 또는 특정 기법으로 검출된 씨드로부터 시작하여 그 주변 픽셀의 특성 값을 계산하고, 계산된 특성값이 현재 분할된 영역의 특성값과 비슷한 범주라고 판단되면 해당 픽셀을 분할 영역에 포함시킨다. 이 과정을 더 이상의 추가되는 픽셀이 없을 때까지 반복하면 최종적으로 분할된 영상을 얻을 수 있다.

특성값을 결정하는 방법은 목적에 따라 여러 가지가 있는데, 일반적으로 많이 쓰이는 방법에는 밝기값(intensity), 경사도(gradient) [7] 가 있고 최근에는 텍스처 특성값을 사용하기도 한다. 다음 식은 SRG의 단계별 영역을 일반식으로 나타낸 것이다.

$$R_i = \{p | F_{low} \le F(N(p)) \le F_{high}, p \in R_{i-1}\} \cup R_{i-1} \quad (1)$$

R_i : i 번째 단계에서 분할된 영역

$N(p)$: p의 주변 픽셀

$F(x)$: x의 특성값

F_{low} : 하한값

F_{high} : 상한값

2.2. 영역 포함 결정 방식

식 (1)에서 분할된 영역에 포함시키기 위한 조건인 값을 정하는 방법은 여러 가지가 있다. 가장 간단한 방법으로 두 가지가 있는데, 첫 번째는 고정적인 상한 값과 하한 값을 정해주는 방법이 있다.

$$C_{Low} < F(x) < C_{High} \tag{2}$$

C_{Low} : 하한 기준값

C_{High} : 상한 기준값

$F(x)$: 확장할 곳의 픽셀(pixel)의 특성값(feature value)

그런데 이 방법은 분할하려는 영역이 다른 영역과 확연히 다른 특성값을 가지고 있고 실험자가 그 특성값을 정확히 알고 있는 경우에 정확한 분할이 가능하다. 하지만 이런 경우는 거의 존재하지 않기 때문에 효율적이지 못하다.

두 번째로 특성값들의 평균값을 이용하는 경우가 있다.

$$\left| Average(i) - F(x') \right| < C_{Avg} \tag{3}$$

C_{Avg} : 평균 기준값

$Average(i)$: 현재까지 영역에 포함된 픽셀의 특성값의 평균

현재 분할된 영역의 특성 값들의 평균값과 포함시킬 픽셀의 특성 값의 차가 어떤 범위 이내에 있을 경우 포함시키는 것이다. 관심이 가는 영역을 분할하려는 경우 그 영역의 특성값들은 비슷한 값을 가질 가능성이

높다. 그래서 어느 정도 오차 범위 내의 값을 추출하게 할 경우 관심 영역을 분할할 가능성이 높다. 관심 영역의 특성값 분포를 정확히 알지 못하는 경우 첫 번째 방법보다 유용하게 사용 가능하다.

그리고 포함시키기 위해 고려하는 픽셀의 주변 픽셀들의 밝기 경사도(intensity gradient)값들의 평균과 상한 값과 하한 값에 적용할 변수를 이용하여 실시간마다 포함 조건의 상한 값과 하한 값을 변화시키는 방법 [8], 표준 편차를 이용하는 방법 [9] 등이 있다.

2.3 다중 SRP 서브 시스템

SRP는 변경 가능한 기본 구조(reconfigurable fabric)를 포함하는 고성능 프로세서로서, 다중 SRP 서브 시스템은 SRP 코어를 이용하여 다양한 코어간 통신과 메모리 구조를 고려하여 설계한 다중 코어 아키텍쳐(Multi/Many-Core Architecture)이다.

멀티 코어 시스템(Multi-core system)의 구조는 주로 성능, 소비전력, 프로그래밍 모델, 확장성 등을 고려하여 결정하게 되며, 이들은 코어들 사이의 통신 방법과 메모리 시스템의 구조에 따라 크게 좌우된다. 통신 방법은 크게 전역적 공유 메모리(globally shared memory)를 통한 방법과 메시지 패싱(message passing) 방법으로 나누어 생각할 수 있고, 메모리 시스템의 구조는 중앙 메모리(centralized memory) 구조와 분산 메모리(distributed memory) 구조로 나누어 생각할 수 있다.

3. 다중 SRP 서브시스템을 이용한 영역 성장 기법 개발

3.1. 다중 코어 환경에서의 영역 성장 기법

본 논문에서는 하나의 씨앗으로부터 원하는 영역을 분할하는 영역 성장 기법을 병렬화하는 방법을 제안한다. 효과적인 분할을 위해 논문 [2]에서 텍스처 특성값을 계산하기 위해 제시한 삼차원 Semi-Variogram 을 개발 및 가속하였다. 기존에 분할된 영역에 포함시킬 때의 비교방식은 현재 분할된 영역의 특성값의 평균과 검사한 복셀의 특성값의 차이를 보고 판단하도록 하였다.

병렬화된 영역 성장 기법의 경우 기본적으로 단일 코어 환경에서의 알고리즘을 유지하되, 모든 코어가 최대한 병렬적으로 작동하게 하기 위해 전파 전략을 사용하였다. 전파 전략이란 어떤 코어가 인접 픽셀에 대한 포함 여부를 결정한 후에, 자신의 큐(queue)에 넣지(enqueue) 않고, 통신을 통해 다른 코어에 넣도록 하여 작업을 분배할 때 사용하는 방식을 말한다. 이 때, 서로 다른 코어가 동일한 코어의 큐에 동시에 넣을 가능성도 있기 때문에 잠금(lock)을 사용하여 일관성을 갖도록 하였다.

```
...
if(RP_ID == 0)
{
   enqueue(queue, seed);
   flag[seed] = INCLUDED | VISITED;
}
```

```
while(1)
{
  if (queue is empty)
  {
    if(every core has an empty queue)
      break;
  }

  item = dequeue(queue);
  for all neighbors of item
  {
    //   calculate texture feature
    ...
    if(feature   of the tested neighbor pixel is OK)
    {
      next_queue = GetNextQueue();
      Lock(next_queue);
      enqueue(next_queue, neighbor);
      Unlock(next_queue);
      flag[neighbor] =   INCLUDED;
    }
  }
}
```

코드 3-1. 다중 코어 환경에서의 씨드 영역 성장 기법 구현

다른 코어에 작업을 전파할 때, 일반적으로 가장 직관적인 방법으로 그림 3-1과 같은 순환 순서 방법을 사용할 수 있지만, 이는 초기에 작업을 전파할 때 느려질 수도 있기 때문에 그림 3-2와 같은 전파 전략을 사용하였다.

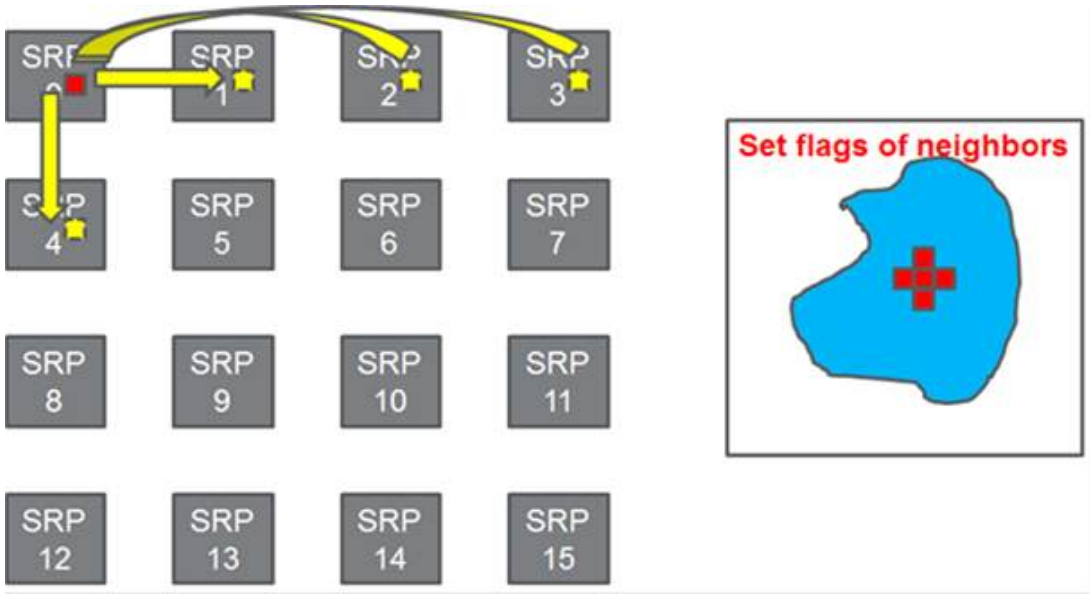

그림 3-1. 순환 순서 방식 작업 전파 전략

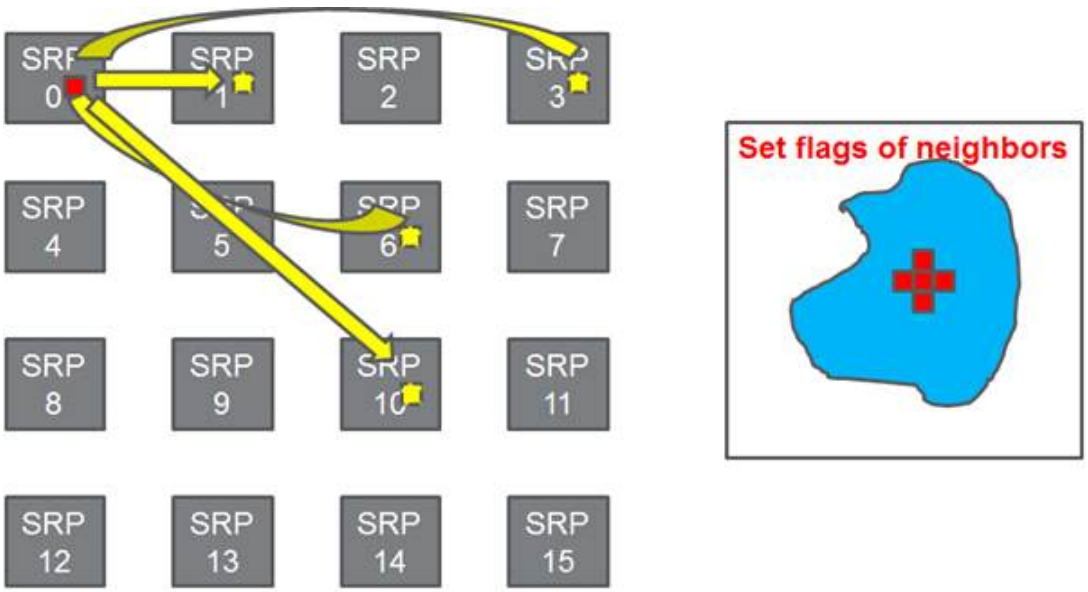

그림 3-2. 제안된 작업 전파 전략

```
propa_core = (propa_core + inc) % (num_of_cores)
inc = inc + 1;
```

제안된 작업 전파 전략을 이용한 코어 0의 전파 순서는 다음과 같다.

1, 3, 6, 10, 15, 5, 12, 4, 13, 7, 2, 14, 11, 9, 8, …

16회 전파될 동안 동일한 코어가 중복되지 않으며 16회 이후에는 8부터 시작하는 것과 동일한 순서로 전파된다. 다른 코어도 위의 순서와 같은 경향으로 전파한다.

3.2. 단위 데이터 전송 (Block data transfer before texture feature calculation)

텍스처 특성값을 계산할 때, 좁은 특정 영역 안에서 계산을 반복하게 되므로, 특정 영역 안의 픽셀에 반복적으로 접근하게 된다. 일반적으로 볼륨 데이터는 용량이 크기 때문에 DRAM에 적재하는데, 이에 대한 접근은 대기 시간(latency)이 크기 때문에 성능 저하가 발생할 수밖에 없다. 이러한 문제를 개선하기 위해 본 논문에서는 텍스처 특성값을 계산하기 전에 필요한 특정 영역을 미리 해당 코어로 가져온 후에 텍스처 특성값을 계산하도록 하였다. 이렇게 할 경우 텍스처 특성값을 얻는 연산을 수행할 때마다 DRAM에 있는 볼륨 데이터를 반복해서 읽지 않아도 되기 때문에 성능향상을 기대할 수 있다. 이는 단일 코어에도 적용이 가

능한 부분으로 정확한 비교를 위해 단일 코어 상의 구현에도 적용을 하였다.

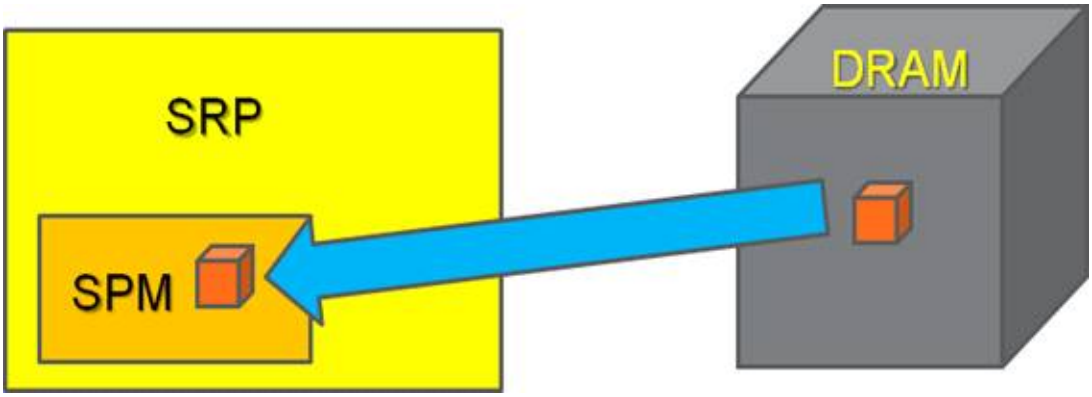

그림 3-3. 단위 데이터 이전

3.3. 다중 코어 환경에서의 종료 조건

단일 코어 환경에서는 분할될 가능성이 있는 픽셀의 리스트를 하나의 큐로 모두 관리하기 때문에 큐가 비어있으면 작업이 종료되도록 하였다. 하지만 다중 코어 환경에서는, 현재 코어의 큐가 비어있다고 해도 다른 코어의 큐가 비어있지 않으면 작업을 종료해서는 안된다. 이를 해결하기 위해, 현재 코어의 큐가 비어있을 경우 다른 코어의 큐가 비어있는지 검사를 하는 추가적인 수행을 하도록 하였다.

4. 실험 결과 및 분석

모든 실험은 다중 SRP 서브시스템 시뮬레이터(mRPsim : Simulator for multi Samsung reconfigurable processor subsystem)에서 수행하였다. 이 시뮬레이터는 프로그램을 단일 SRP로 실행하였을 경우와 다중 SRP로 실행하였을 경우로 나누어서 측정이 가능하다. 그리고 프로그램을 실행하였을 때 결과로 각각의 프로세서의 사이클 수와 모든 프로세서를 이용한 전체적인 사이클 수를 결과로 보여준다. 결과로 나타나는 사이클 수는 시뮬레이터로 돌린 것이므로 컴퓨터의 성능과 무관하게 항상 같은 결과값을 얻을 수 있다. 시뮬레이터는 CentOS-5.5를 기반 운영체제로 이용한다.

시뮬레이터에서 가정하고 있는 코어의 수는 16개로 DRAM과 코어의 배치는 그림 4-1과 같다.

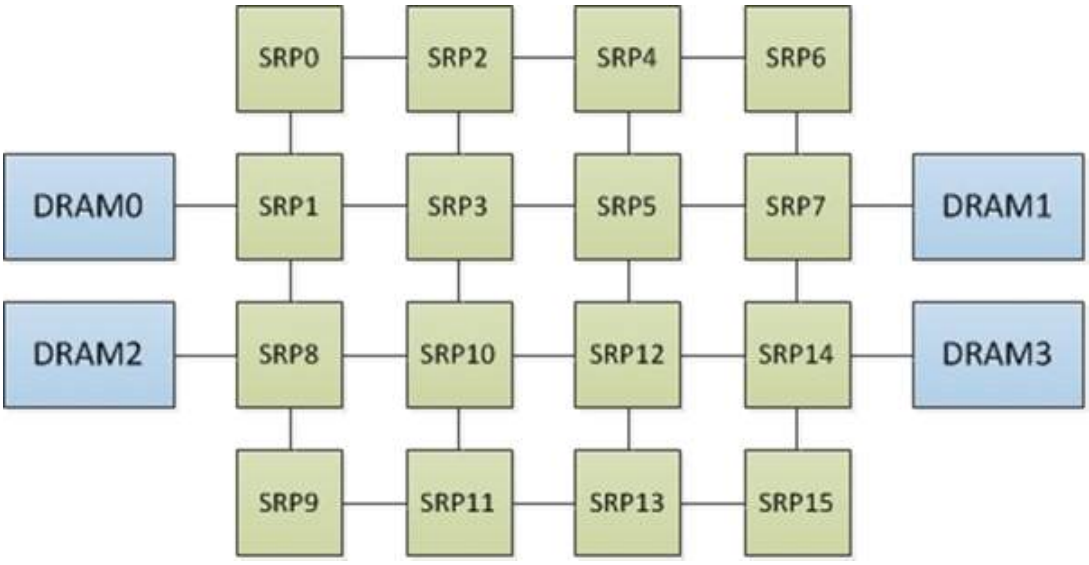

그림 4-1. 다중 SRP의 DRAM과 SRP 코어의 배치도

모든 코어는 병렬적으로 작업을 처리하지만, 처리 도중 잠금을 걸고 각각의 코어에 있는 데이터를 서로 읽고 쓸 수 있다. 본 논문에서는 다중 SRP 서브 시스템 시뮬레이터가 아토믹(atomic) 연산을 지원하지 않아 스핀 락(spin lock)을 사용하였다.

텍스처 특성값을 계산하기 위해 삼차원 Semi-Variogram을 이용하였으며 마스크(mask)크기는 5x5x5를 이용하였다. 다중 SRP 코어상의 실험은 단일코어 대비 시뮬레이션 시간이 약 20000배정도 걸리는 관계로, 측정 실험에서는 볼륨 중앙에 정육면체의 물체가 존재하는 데이터를 자체적으로 제작하여 사용하였다. 크기는 64x64x64인 8비트 그레이 영상이다. 입력 데이터와 데이터의 분할 결과를 저장하는 플래그(flag) 데이터는 다중 코어의 경우 코어 간의 공유가 필요하기 때문에 공유 메모리 영역에 할당을 하였고, 정확한 비교를 위해 단일 코어 환경에서도 동일하게 할당하였다.

4.1. 단일 코어 환경 실험 결과

표 4-1은 단일 코어에서 최적화에 따른 실험 결과이다. 실험에서 알 수 있듯이, 필요한 만큼의 데이터를 텍스처 특성값 계산 전에 DRAM으로부터 가져오는 것이 효율적으로 성능을 향상시킨 것을 볼 수 있다. 비록 이를 위한 코어 사이클(core cycle)은 약간 증가하였지만 전체적인 성능향상을 위한 추가 작업으로 간주할 만하다. 최종적으로 최적화를 진행하였을 때 전체 성능이 약 2.86배 정도의 성능 향상을 보였다.

표 4-1. 최적화에 따른 실험 결과

최적화	global		core	
	cycle	speed-up	cycle	speed-up
Basic	417,093,248	1	61,478,938	1
Block transfer	222,049,838	1.88	82,643,972	0.74
Optimization	145,601,849	2.86	33,518,234	1.83

4.2 다중 코어 환경 실험 결과

표 4-2는 다중 코어 환경에서 실험한 결과이다. 실험 결과, 약 10.98배의 성능 향상을 보였다. 예상했던 것보다 낮은 수치가 나왔는데, 이는 코어마다 볼륨 데이터로의 접근 시간이 다르기 때문인 것으로 파악된다. 실험 당시 볼륨 데이터가 DRAM0에 위치하고 있었기 때문에 볼륨 데이

터가 저장된 DRAM0와 각 코어의 거리가 달라 접근 시간에 차이가 날 수밖에 없었고, 이것이 성능 향상에 영향을 주었다고 생각한다. 그림 4-1을 살펴보았을 때, 구조상 코어0 과 코어15 에서 한 번의 텍스처 특성값을 구하기 위해 필요한 사이클 수는 크게 다를 것이고, 이러한 이유로 코어 수에 따른 성능 향상은 제한적일 수밖에 없다. 코어 사이클의 경우 코어에 따라 크게는 1.99배 정도의 차이가 나고 있는데, 이는 위의 생각을 뒷받침해주고 있다.

표 4-2. 다중 코어 환경에서 실험 결과

(하나의 데이터를 DRAM0에 적재)

core id	global	core
single	145,601,849	33,517,434
core0	13,260,418	3,046,602
core1	13,260,434	3,743,283
core2	13,260,447	2,509,434
core3	13,260,412	2,989,208
core4	13,260,412	2,214,164
core5	13,260,404	2,486,630
core6	13,260,402	2,063,301
core7	13,260,402	2,284,551
core8	13,260,408	3,015,739
core9	13,260,468	2,586,829
core10	13,260,402	2,616,190
core11	13,260,455	2,140,193
core12	13,260,424	2,159,052
core13	13,260,436	1,980,478
core14	13,260,402	1,926,335
core15	13,260,429	1,882,861
SUM(0~15)	212,166,755	39,644,850
speed-up	10.98	8.95

표 4-3은 각 코어의 텍스처 특성값 계산 횟수(texture feature count)를 나타낸 결과이다. 그 결과 연산량이 특정 코어에 몰린 것을 볼 수 있다. 이는 앞서 기술한대로, 볼륨 데이터가 저장되어 있는 DRAM0와 각 코어 간의 거리가 다르기 때문으로 보인다. DRAM0와의 거리가 가장 가까워 볼륨 데이터로의 접근이 가장 빠른 코어1이 텍스처 특성값 연산에 유리하므로 연산 횟수가 가장 많았고, 마찬가지로 DRAM0와의 거리가 가장 먼 코어15 의 경우 연산 횟수가 가장 적었다.

표 4-3. 코어 당 텍스처 특성값 계산 횟수

Core ID	0	1	2	3	4	5	6	7
Texture feature count	2112	2800	1721	2161	1446	1740	1251	1465
Core ID	8	9	10	11	12	13	14	15
Texture feature count	2117	1694	1726	1434	1448	1240	1246	1077

표 4-4는 모든 DRAM에 동일한 볼륨 데이터를 적재하고 각 코어에서 가장 가까운 DRAM에 접근하도록 코드를 수정한 후 실험한 결과이다. 실험 결과, 약 13.93배의 성능 향상이 나타난 것을 볼 수 있었다. 이는 각 코어가 볼륨 데이터에 접근하는 시간이 전체적으로 감소하여 그에 따라 모든 코어가 이전보다 더 균등하게 작업을 수행하게 되었기 때문으로 분석된다.

표 4-4. 다중 코어 환경에서 실험 결과

(모든 DRAM에 동일한 볼륨 데이터 적재)

core id	global	core
single	145,601,849	33,517,434
core0	10,453,963	2,513,782
core1	10,453,934	3,073,788
core2	10,453,947	2,026,451
core3	10,453,959	2,532,288
core4	10,453,948	2,114,097
core5	10,453,951	2,489,660
core6	10,453,950	2,496,248
core7	10,453,934	2,924,801
core8	10,453,988	2,924,229
core9	10,453,987	2,418,767
core10	10,453,934	2,414,145
core11	10,453,937	1,930,974
core12	10,453,935	2,375,417
core13	10,453,952	1,948,258
core14	10,453,936	2,912,622
core15	10,453,936	2,235,407
SUM(0~15)	167,263,191	39,330,934
speed-up	13.92787604	10.90427642

4.3 CPU와의 결과 비교

추가적으로 CPU에서 영역 성장 기법을 적용하여 비교하는 실험을 하였다. 객관적인 비교를 위해서 실제 인체 데이터(256x256x102, 16비트)를 이용하여 실험하였다. 실험에 사용된 CPU는 Intel Core2-i7 @ 3.40 GHz, DDR3 1333GHz로 각각 현재 최신 장치이다. 개발 환경은 Visual Studio 2010을 사용하였다. CPU 상에서 구현된 내용은 본 연구에서 개발한 단일 코어 mrpsim 구현 내용과 동일하다. SRP의 성능은 현재 시뮬레이터 상의 사이클 외에는 객관적인 수치가 없으므로 코어 동작 클럭을 1GHz로 가정하여 추산하였다.

표 4-5는 영역 성장 기법을 각각 CPU, SRP 상에서 수행한 결과이다. 표에서 보듯이, SRP가 CPU와 비슷한 성능을 보였다. 하지만 하드웨어 사이즈 등의 차이를 고려하면 좋은 성능을 보여준 것으로 분석된다. 또한 비교 대상이 현재 최신 장치라는 점에서 다중 코어 기반의 SRP가 효율적으로 영역 성장 기법을 가속하였다고 볼 수 있다.

표 4-5. CPU/SRP 성능 비교

	CPU	SRP
시간(초)	0.951	0.967

그림 4-2는 각각의 환경에서 분할한 결과이다. 실제 분할 결과는 볼륨 데이터이지만 편의상 특정 슬라이스 한 장씩 비교하였다. 그 결과, 육안으로는 큰 차이가 관찰되지 않았다. 그래서 보다 객관적인 비교를 위해 CPU의 결과를 기준으로 하여 차영상(difference image)을 만들어 살펴보았다. 그림 4-3은 그 결과로 이 역시 큰 차이가 관찰되지 않았다. 볼륨

데이터 전체에 대해서 CPU의 결과와 비교하여 일치하지 않은 복셀의 수를 세어본 결과, SRP의 경우 295개밖에 발견되지 않았다. 이를 통해 제안된 기법이 효과적임을 알 수 있다.

(a) CPU (b) SRP

그림 4-2. CPU, SRP 기반 분할 결과 영상

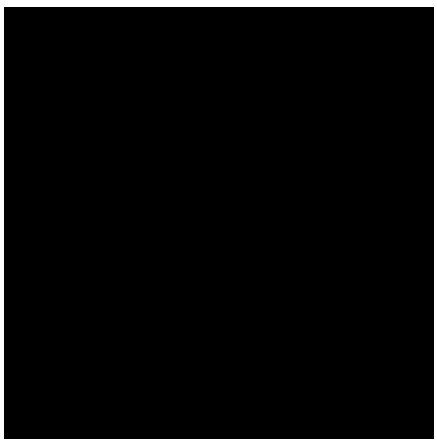

그림 4-3. CPU-SRP 차영상

5. 결론 및 향후 연구 방향

5.1. 결과 정리 및 요약

영역 성장 기법은 문제 영역의 크기가 정해져 있지 않고, 시간에 따라 변하기 때문에 일반적인 데이터 병렬 처리 방법으로는 적용이 용이하지 않다. 본 논문에서는 이러한 영역 성장 기법을 다중 SRP 서브 시스템에 적용하여, 단일 코어 환경에서의 최적화된 성능 대비 약 11배 가까운 성능 향상을 볼 수 있었다. 최종적으로는 볼륨 데이터가 존재하는 DRAM과 각 코어 간의 거리가 달라 모든 코어가 수행하는 작업량이 균등하지 못했기에 더 큰 성능 향상을 보이지 못했다.

5.2. 성능 향상을 위한 향후 연구 방향

현재 실험 결과상으로는 작업 분배가 더 효율적으로 이루어지면 추가적인 성능 향상이 있을 수 있음을 예상할 수 있다. 근본적으로 병렬 프로그래밍 환경에서 필수적인 아토믹(atomic) 연산과 배리어(barrier)가 존재하지 않아 이를 소프트웨어적으로 구현하는데 따른 오버헤드가 발생하긴 하였다. 하지만 이를 고려하더라도 병렬 프로그래밍의 특성 상 모든 코어에 고르게 작업을 분배하는 것이 성능 향상의 극대화를 위해 꼭 필요할 것으로 보인다. 그렇게 하기 위해 코어마다 가중치를 두는 방식이나, DRAM에 가까운 소수의 코어를 상위 구조로 삼고 코어에서 먼 다수의 코어를 하위 구조로 삼아 트리 형태로 일을 전파하는 방식 등의 각각

의 코어에 작업의 분배를 고르게 하는 방식을 생각해 볼 수도 있다.

참고문헌

[1] Stefan Schenke, Burkhard C.Wünsche and Joachim Denzler, "GPU-Based Volume Segmentation," Proceedings of IVCNZ 2005, 2005.

[2] Jie Wu, Skip Poehlman, Michael D.Noseworthy, Markad V.Kamath "Texture feature based automated seeded region growing in abdominal MRI segmentation," J.Biomedical Science and Engineering, 2, 1-8, 2009.

[3] R. Adams and L. Bischof, "Seeded Region Growing," IEEE Trans. PAMI, vol. 16, no. 6, pp. 641-647, 1994.

[4] N. A. Mat-Isa, M. Y. Mashor & N. H. Othman, (2005) "Seeded Region Growing Features Extraction Algorithm; Its Potential Use in Improving Screening for Cervical Cancer," International Journal of the Computer, the Internet and Management. (ISSN No: 0858-7027). Vol. 13. No. 1. January-April, 2005.

[5] Y. Tuduki, K. Murase, M. Izumida, H. Miki, K. Kikuchi, K. Murakami & J. Ikezoe (2000). "Automated Seeded Region Growing Algorithm for Extraction of Cerebral Blood Vessels from Magnetic Resonance Angiographic Data," Proceedings of The 22nd Annual International. Conference of the IEEE Engineering in Medicine and Biology Society. 3. 1756-1759, 2000.

[6] P. A. Venkatachalam, U. K.Ngah, A. F. M. Hani& A. Y. M. Shakaff, (2002). "Seed Based Region Growing Technique in Breast Cancer Detection and Embedded Expert System," Proceedings of International Conference on Artificial Intelligence in Engineering and Technology. 464-469, 2002.

[7]] Nair T. R. G. Rai, G. N. H. and. "Gradient based seeded region grow method for ct angiographic image segmentation," CoRR, abs/1001.3735, 2010.

[8] T. Grenier, C. Revol-Muller, N. Costes, M. Janier, and G. Gimenez, "3D robust adaptive region growing for segmenting [18F] fluoride ion PET images," in IEEE Nuclear Science Symposium Conference Record, San Diego, CA, United States, 2007, pp. 2644-2648, 2007.

[9] R. Pohle and K. D. Toennies, "Segmentation of medical images using adaptive region growing," presented at Medical Imaging 2001 Image Processing, Feb 19-22 2001, San Diego, CA, 2001.

[10] J. Choi, S. Kim,H. Han. "Accelerating loops for coarse grained reconfigurable architectures using instruction extensions," in Proceedings of the 2011 ACM Symposium on Research in Applied Computation. 2011. Miami, Florida: ACM.314-318, 2011.

Abstract

Parallel Seeded Region Growing based on Intercommunicating Multi-Core

Dohyun Kim

School of Computer Science and Engineering

The Graduate School

Seoul National University

Segmenting the region of interest is commonly done before analyzing the images. But, as the degree of precision in medical image equipment elevates, the size of dataset also increases, resulting in lengthy computation time for segmentation. The process can be parallelized in order to reduce the time.

This paper proposes a method to process the seeded region growing of the image segmentation methods, with a parallelized algorithm. It is not trivial to process the region growing with general parallel

processing, because the problem size is often not fixed, and changes over time. This paper presents the acceleration techniques using the feature of multi SRP subsystem, which allows communication among each core. This paper uses other work propagate discipline instead of round robin discipline for fast, and uniform distribution of work to each core. In this scheme, the performance increases about 10.98 times compared to single-core environment.

Keywords : Segmentation, Seeded Region Growing, Parallel Processing, multi Samsung reconfigurable processor subsystem

Student Number : 2011-20801