JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 20, No. 10 : 1919~1926 Oct. 2016

전달 루틴의 병렬화를 통한 SAT 알고리즘의 GPGPU 가속화

강형주*

GPGPU Acceleration of SAT Algorithm with Propagation Routine Parallelization

Hyeong-Ju Kang*

School of Computer Science and Engineering, Korea University of Technology and Education, Cheonan 31253, Korea

요 약

대량의 데이터를 병렬적으로 처리할 수 있는 General-Purpose Graphics Processing Unit(GPGPU)가 최근 많은 분야에서 적용되고 있으며, 이는 전자 설계 자동화 분야에서도 예외가 아니다. SAT 알고리즘은 다양한 전자 설계 자동화 문제에 적용되는 대표적인 알고리즘 중 하나이다. GPGPU를 이용해서 SAT 알고리즘을 가속화하기 위해 노력이 이루어져 왔으나, SAT 알고리즘 자체의 특성으로 인해 병렬화에 어려움이 있어왔다. 이 논문에서는 SAT 알고리즘의 내부 과정 중 비교적 병렬화가 용이한 전달 루틴을 병렬화함으로써 GPGPU 가속화를 적용하였다. 전달 루틴이 희소 행렬의 곱셈과 유사한 점에 착안하여 데이터 구조를 구성하고 이에 맞추어서 병렬적인 전달 루틴을 작성하였다. 병렬적으로 동작하는 쓰레드들 사이의 데이터 손실을 방지하기 위해 아토믹(atomic) 연산을 이용하였다. 벤치마크 SAT 문제들에 대해 기존의 GPGPU 기반 SAT solver에 비해 성능이 10배 이상 향상되었음을 확인하였다.

ABSTRACT

Because of the enormous processing ability, General-Purpose Graphics Processing Unit(GPGPU) has been applied to many fields including electronics design automation. The SAT algorithm is one of the core algorithm in many electronics design automation tools. There has been some efforts to apply GPGPU to the SAT algorithm, but it is difficult to parallelize the SAT algorithm because of its characteristics. In this paper, I applied GPGPU to the SAT algorithm by parallelizing the propagation routine that is relatively suitable to parallel processing. On the basis of the similarity of the propagation routine to the sparse matrix multiplication, the data structure for the SAT problem is constituted, and the parallel propagation routine is described. To prevent data loss between paralllel threads, atomic operations are exploited. The experimental results for some benchmark SAT problems show that the proposed algorithm is superior to the previous GPGPU-based SAT solver.

키워드 : GPGPU, SAT 알고리즘, 병렬화, 알고리즘 가속화, 회로 설계 자동화

Key word : GPGPU, SAT Algorithm, Parallelization, Algorithm Acceleration, Electronics Design Automation

Received 11 August 2016, **Revised** 31 August 2016, **Accepted** 08 September 2016

* **Corresponding Author** Hyeong-Ju Kang(E-mail:hjkang@koreatech.ac.kr, Tel:+82-41-560-1420)
School of Computer Science and Engineering, Korea University of Technology and Education, Cheonan 31253, Korea

Open Access http://dx.doi.org/10.6109/jkiice.2016.20.10.1919 print ISSN: 2234-4772 online ISSN: 2288-4165

ⓒThis is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License(http://creativecommons.org/li-censes/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.
Copyright Ⓒ The Korea Institute of Information and Communication Engineering.

Ⅰ. 서 론

단일 코어 CPU 성능의 발전이 더디게 되면서, 최근에는 General Purpose Graphics Processing Unit (GPGPU)가 고성능 컴퓨팅에서 점점 중요한 역할을 하고 있다. 일반적인 멀티 코어 CPU가 태스크 수준에서의 병렬화를 채택하고 있는 반면에 GPGPU에서는 데이터 수준의 병렬 프로그래밍 모델을 사용하고 있다. GPGPU는 수천 수만 개의 fine-grain 쓰레드들을 동시에 실행하며, 이들 쓰레드들은 같은 프로그램을 서로 다른 데이터에서 실행한다.

이런 흐름에서 전자 설계 자동화(EDA, Electronic Design Automation) 분야의 특정한 문제들에 대해서는 이미 GPU를 적용하는 것이 시도되었다[1,2]. 그러나 전자 설계 자동화 분야의 알고리즘에서는 불규칙적인 형태의 데이터를 다루어야 하는 경우가 많고 이러한 불규칙적인 형태의 데이터는 GPU의 병렬 연산을 적용하기에 어려움이 있어서 아직 많은 연구가 필요한 상황이다.

전자 설계 자동화 분야의 대표적인 알고리즘 중 하나인 Satisfiability(SAT) 알고리즘은 주어진 불 대수식이 참이 되게 하는 변수들의 값, 즉 해가 존재하는지를 판별하는 알고리즘이다. 대표적인 NP-complete 문제이고 전자 설계 자동화 알고리즘에서 자주 사용되므로, 많은 연구가 이루어져 왔다.

SAT 알고리즘은 크게 complete SAT 알고리즘과 incomplete SAT 알고리즘으로 나뉘며, incomplete SAT 알고리즘은 해가 존재하는 문제에서는 그 해를 빨리 찾는 반면에 해가 존재하지 않는다는 것은 증명하지 못하고, complete SAT 알고리즘은 해가 존재하지 않음도 증명할 수 있는 알고리즘이다. 대부분의 complete SAT 알고리즘은 branch-and-bound 알고리즘과 비슷한 형태의 DPLL 알고리즘에 기반하고 있으며, 초기의 DPLL 알고리즘에 기반을 둔 SAT solver에서 시작하여 GRASP[3], Chaff[4], Minisat[5] 등을 거쳐 최근에는 Glucose[6] 등의 SAT solver들이 제안되었다.

단일 코어를 위해 개발된 SAT 알고리즘들은 다중 코어 환경을 위해 확장되기도 했다. 여러 시도 끝에 최근에 많이 사용되고 있는 것은 같은 SAT solver를 여러 파라메터로 실행하는 프로파일 기반 병렬 알고리즘이다[7]. 그러나 이들 병렬 SAT 알고리즘들은 여러 개의 SAT solver를 병렬적으로 실행하는 것과 같으므로 하나의 SAT 알고리즘 내부에서 병렬적으로 동작하는 것과는 거리가 멀다. 하나의 SAT 알고리즘 내부를 병렬화하는 것도 연구가 되었으나 주로 2~4개 정도의 다중 코어 환경을 가정한 것이었고 큰 효과를 거두지 못했었다.

SAT solver의 성능을 향상시키기 위해 수백 개 이상의 연산 유닛을 가지는 GPGPU를 적용하는 노력도 이루어져 왔다[8,9]. 그러나 병렬화에 맞지 않는 DPLL 알고리즘의 내부 루틴들과 SAT 문제 자체가 가지고 있는 여러 성질들로 인해 GPGPU의 적용이 어려웠었다. SAT 문제는 불규칙적인 데이터 형태를 가지고 있으며, 적은 양의 데이터가 여러 차례에 걸쳐 전달되는 형태로 진행된다. 그리고 중간에 변수의 값을 결정하거나 충돌 상황을 해결하는 등의 병렬화가 되지 않는 부분들을 지속적으로 수행해야 한다.

본 연구에서는 SAT 알고리즘에 GPGPU를 적용하는 방법에 대해 연구하였다. SAT 알고리즘 전체를 GPGPU로 옮기는 대신, 기존의 SAT 알고리즘에서 가장 시간이 오래 걸리면서도 병렬화에 비교적 용이한 전달 루틴을 GPGPU로 포팅하고 나머지 부분은 기존의 알고리즘을 이용하였다. 이를 통해 기존 알고리즘의 우수성을 이용하면서도 전달 루틴은 빠른 SAT solver를 구성할 수 있었다.

이 논문은 다음과 같이 구성되어 있다. 2장에서는 SAT 알고리즘에 대해 설명하고, 3장에서는 GPGPU에 SAT 알고리즘을 적용할 때 발생하는 문제점들을 기술한다. 4장에서는 이러한 문제점들을 해결하는 기법들을 제안하고, 5장에서 실험결과를 제시한 뒤, 6장에서 결론을 내린다.

Ⅱ. SAT 알고리즘

SAT 알고리즘은 주어진 불 대수식을 참이 되게 하는 변수들의 값의 조합, 해가 존재하는지 판별하는 알고리즘이다. SAT 알고리즘에 주어지는 불 대수식은 그림 1과 같은 Conjunctive Normal Form(CNF) 형태로 표현된다.

CNF 형태에서 하나의 불 대수식은 여러 절(clause)들의 AND로 표현되고, 하나의 절은 여러 문자(literal)들의 OR로 표현된다. 문자는 양 문자(positive literal)와

음 문자(negative literal)가 있으며, 양 문자는 불 변수 자체이고, 음 문자는 불 변수의 NOT이다. 이렇게 표현된 불 대수식이 참이 되려면 모든 절들의 값이 참, 즉 1이 되어야 하며, 만일 하나의 절이라도 값이 0이면 전체 불 대수식의 값도 0이 된다.

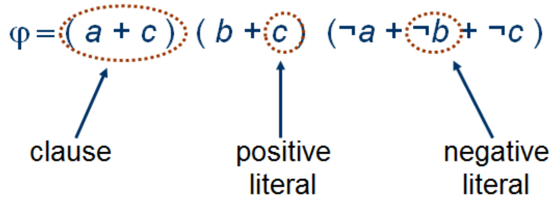

Fig. 1 Boole equation in CNF

이러한 문제를 푸는 SAT 알고리즘은 기본적으로 그림 2와 같은 DPLL 알고리즘에 기반하고 있다. 그림의 6번 줄에 있는 선택 결정 루틴(decide())은 아직 값이 정해지지 않은 변수들 중 하나를 뽑아서 임의의 값을 지정하는 루틴이다. 이 루틴을 통해 어떤 변수의 값이 정해지면 2번 줄의 전달 루틴(propagate())을 거치게 된다. 전달 루틴에서는 기존에 값이 결정된 변수들로 부터 다른 변수의 값을 알아내는 단계이다. 만일 어떤 절에서 한 문자의 값은 아직 정해지지 않았고 그 것을 제외한 다른 모든 문자들의 값이 0이면(이러한 절을 유닛 절(unit clause)라고 부른다.), 그 절의 값이 1이 되기 위해서는 그 정해지지 않은 문자의 값이 1이 되어야 한다. 따라서 그 문자에 해당하는 변수의 값을 알아 낼 수 있는 것이다. 이렇게 유닛 절을 판별하고 그에 따라 변수들의 값을 알아내는 과정이 전달 루틴이다.

이렇게 선택 결정 루틴과 전달 루틴이 진행되다가 값이 0이 되는 절(충돌 절, conflicting clause)을 발견하면 9번줄의 충돌 해석 루틴(analyze())을 실행한 뒤 13번 줄과 같이 되추적(bracktrack)을 한다. 되추적에서는 이전의 선택 결정 과정으로 돌아가서 결정했던 변수의 값을 반대로 되돌리며, 만일 그 변수에 대해 0과 1을 모두 시도했었다면 그 이전의 선택 결정 과정으로 되돌린다.

원래의 DPLL 알고리즘에서는 9번 줄의 충돌 해석 루틴이 없었으나, GRASP라는 SAT solver부터 추가되었다[3]. 이 충돌 해석 루틴에서는 충돌 절의 원인을 분석하여 다시 비슷한 충돌이 발생하는 상황을 방지하도록 충돌 방지 절(conflict clause)을 추가한다.

GRASP에서 충돌 해석 루틴이 제안된 이후로, Chaff라는 SAT solver에서는 watching 알고리즘을 이용하여 전달 루틴의 속도를 높였다[4]. 그 다음으로 기존의 연구 결과들을 집약하면서도 확장가능한 간단한 SAT solver인 MiniSAT이 제안되면서[5] 현대 대부분의 SAT solver들은 MiniSAT에 기반하고 있다. 최근에는 Glucose라는 SAT solver가 등장하였으며, 이 solver는 MiniSAT에 효율적인 충돌 방지 절 관리 기법을 적용한 것이다.

```
01: loop
02:   propagate()              // propagate unit clauses
03:   if not conflict then
04:     if all variables assigned then
05:       return SATISFIABLE
06:     else
07:       decide()             // pick a new variable and assign it
08:   else
09:     analyze()              // analyze conflict and add a conflict clause
10:     if top-level conflict found then
11:       return UNSATISFIABLE
12:     else
13:       backtrack()          // undo assignments until conflict clause is unit
```

Fig. 2 Basic SAT algorithm[5]

Ⅲ. GPGPU와 SAT 알고리즘

이 장에서는 GPGPU에 대해 설명하고 SAT 알고리즘에 적용할 때의 문제점에 대해 기술한다.

3.1. GPGPU

Graphics Processing Unit(GPU)는 본래 컴퓨터에서 3D 그래픽 처리 작업을 원활히 하기 위해 개발되었으며, 다수의 작고 단순한 연산 회로를 가지고 있어서 단순한 연산을 병렬적으로 처리하기에 적합한 구조를 가지고 있다. 높은 연산 능력으로 인해 3D 그래픽 이외의 분야에서도 적용하려는 노력이 이루어져 왔고, 이를 위해 GPGPU라는 개념이 개발되었다.

대표적으로 NVIDIA사의 CUDA와 OpenCL이 있다. GUP의 예로 NVIDIA사의 GeForce GTX 980은 Maxwell 구조를 채택한 GPU로써 네 개의 GPC (Graphics Processing Clusters)로 이루어져 있고, 한 개의 GPC에는 4개의 SMM(Maxwell Streaming Multiprocessor)가 있으며, 각 SMM들은 128개의 CUDA 코어로 이루어져 있다[10]. 그러므로 GTX 980에는 2048개의 CUDA 코어가 들어 있고 최대 2048개의 연산을 동시에 수행할 수 있다.

3.2. GPGPU에서의 희소 행렬 연산

GPGPU에 관련하여 많이 연구되는 주제 중 하나가 희소 행렬(sparse matrix)의 연산, 특히 행렬-벡터 곱셈(SpMV, Sparse Matrix Vector Multiplication)에 관한 것이다. 희소 행렬은 행렬의 대부분의 요소들의 값이 0인 행렬을 말한다. 이 논문에서 제안하는 기법과 연관이 깊어서 이 절에서 간단히 설명할 것이다.

희소 행렬은 조밀 행렬(dense matrix)에 비해 연산량 자체는 적으나 병렬화에 어려움이 있어서, 병렬화를 위해 많은 연구가 이루어져왔다. 예를 들어 [11]에는 SpMV를 GPGPU에서 수행하는 문제에 대해 다양한 방법을 제시하고 있다. 희소 행렬을 다룰 때 우선 부딪히는 문제는 희소 행렬의 표현이다. 희소 행렬을 조밀 행렬과 같이 모든 요소를 저장하는 방식으로 표현하면 많은 공간을 낭비하게 되므로, 효율적으로 저장하는 여러 가지 방법들이 제시되었는데 표 1이 대표적인 예들이다.

이해를 돕기 위해 $A=\begin{bmatrix}1&7&0&0\\0&2&8&0\\5&0&3&9\\0&6&0&4\end{bmatrix}$ 를 표현하는 예를 제시하였다.

Table. 1 Representation of Sparse Matrix

Format	Example
Diagonal	$data=\begin{bmatrix}.&1&7\\.&2&8\\5&3&9\\6&4&.\end{bmatrix},\ offsets=[-2\ 0\ 1]$
ELLPACK	$data=\begin{bmatrix}1&7&.\\2&8&.\\5&3&9\\6&4&.\end{bmatrix},\ indices=\begin{bmatrix}0&1&.\\1&2&.\\0&2&3\\1&3&.\end{bmatrix}$
COO(Coordinate)	$row=[0\ 0\ 1\ 1\ 2\ 2\ 2\ 3\ 3]$ $col=[0\ 1\ 1\ 2\ 0\ 2\ 3\ 1\ 3]$ $data=[1\ 7\ 2\ 8\ 5\ 3\ 9\ 6\ 4]$
CSR(Compressed Sparse Row)	$ptr=[0\ 2\ 4\ 7\ 9]$ $col=[0\ 1\ 1\ 2\ 0\ 2\ 3\ 1\ 3]$ $data=[1\ 7\ 2\ 8\ 5\ 3\ 9\ 6\ 4]$

이러한 여러 방법들 중에서 희소 행렬의 특성에 따라 적절한 표현법을 고르는 것이 중요하고 이들 각각에 대해서 병렬적으로 처리하는 방법도 역시 서로 다르다. 한 예로, ELLPACK 형식은 각 행별로 반복을 할 때 서로 거의 비슷한 횟수를 반복하므로 병렬적으로 반복처리하기에 편리한 반면에 각 행에 있는 non-zero 요소의 수가 서로 비슷해야 한다는 제약이 있다. 또한 메모리에 저장된 데이터를 병렬적으로 접근하기 위해서는 각 요소들이 메모리의 서로 다른 뱅크에 저장되어야 한다.

3.3. GPGPU와 SAT 알고리즘

GPU의 높은 연산 능력을 이용하여 SAT 알고리즘을 가속화하는 연구가 일부 이루어져 왔으나 그리 성공적이지는 않았다. 논문 [12]에서는 절이 세 개 이하의 문자만 가지는 3-SAT 문제에 대해 GPGPU를 위한 알고리즘을 제안하였으나 다른 SAT solver와의 비교가 없어서 어느 정도 성능 향상을 이루었는지 알 수가 없다. 논문 [8]에서는 3-SAT 문제에 대해 전달 루틴을 GPU에서 수행하는 기법을 도입하였고 논문 [9]에서는 일반적인 SAT 문제에 대해 GPU에서 수행되는 SAT solver를 제시하였으나, 역시 다른 SAT solver와의 비교가 없다.

SAT 알고리즘에서 GPGPU를 이용하기 어려운 문제 중 하나는 SAT 알고리즘을 구성하는 내부 루틴들이 병렬화에 맞지 않다는 점이다. 내부 루틴들 중 선택 결정 루틴은 값이 결정되지 않은 변수들 중 하나를 선택해서 값을 결정해 주는 과정이고, 충돌 해석 루틴은 충돌이 발생한 절로부터 역방향으로 추적하는 과정이다. 이 과정들은 다수의 데이터를 동시에 처리하는 과정이 아니므로 병렬화를 적용하기 어렵다. 또한 이 두 부분은 SAT solver들 사이의 성능차를 만드는 주요 부분이어서 복잡한 기법들이 적용되어 있으므로 병렬화를 더욱 어렵게 하고 있다.

SAT 알고리즘에 GPGPU를 적용하기 어렵게 만드는 또 하나의 이유는 불규칙한 데이터 형태이다. SAT 문제는 많은 수의 절로 이루어져 있는데, 이 절들은 다양한 수의 문자로 구성되어 있다. 이렇게 절들의 길이가 다양함에 따라 CUDA 환경에서 알고리즘을 수행하였을 때, 개개의 코어나 쓰레드가 처리할 데이터의 양이 달라지고, 따라서 병렬화에 따른 장점이 감소한다.

Ⅳ. 제안하는 GPGPU 기반 SAT 알고리즘

본 연구에서는 GPGPU에 기반을 둔 SAT 알고리즘을 제안한다. 기존의 단일 코어 기반 SAT 알고리즘에 대한 성과들을 이용하면서, 시간이 오래 걸리는 전달 루틴을 GPU에서 실행함으로써 기존의 GPGPU 기반 SAT 알고리즘에 비해 높은 성능을 얻고자 한다.

4.1. 기존 직렬 알고리즘과의 융합

제안하는 알고리즘은 MiniSAT을 이용하여 구현되었으며 MiniSAT에서 전달 루틴을 분리하여 CUDA 환경으로 이식하였다. 전달 루틴은 여러 개의 절에 대해 동작하므로 다른 루틴들에 비해 비교적 병렬화가 용이하다. 또한 이렇게 하면 기존에 MiniSAT에 이용되었던 선택 결정 기법이나 충돌 해석 기법을 그대로 이용할 수 있으며, glucose와 같은 많은 수의 현대 SAT solver들이 MiniSAT 기반으로 작성되어 있고 비슷한 전달 루틴을 사용하고 있어서 이러한 SAT solver들로의 적용도 용이하다. 논문 [9]에서도 전달 루틴만을 GPU에서 수행하고 나머지 부분은 CPU에서 수행하는 방법에 대해 논의하고 있으나, 발전된 선택 결정 기법이나 충돌 해석 기법을 이용하지 못했다.

4.2. 제안하는 전달 루틴

SAT 알고리즘에서의 전달 루틴에 GPGPU를 적용하면서 착안한 것은, 이 과정이 SpMV와 많이 유사하다는 것이다. 전달 루틴에서는 불균일한 길이를 갖는 절들에 대해 각 절의 문자에 해당하는 변수의 값을 읽은 뒤 그러한 값들을 조합하여 절의 값을 계산하고 그 것으로부터 전달을 진행한다. SpMV에서도 행렬의 행들이 불

```
01:  propagate()
02:     i = clause index;        // calculated from CUDA thread index
03:     for(pos = ptr[i]; pos < ptr[i+1]; pos++)
04:        literal = literal_array[pos];
05:        if polarity of literal is equal to variable value then
06:           count--;
07:        elif polarity of literal is opposite of variable value then
08:           count++;
09:     if count is equal to clause length then
10:        return conflict;
11:     elif count is equal to clause length–1 then
12:        propagate to unassigned variable;
13:     return;
```

Fig. 3 Proposed propagate routine

```
01: que_addr = atomic_increment(conflict_que[0]);
02: conflict_que[que_addr] = conflicting clause;
```

Fig. 4 Que store using atomic operation

```
01:  que_addr = atomic_increment(variable_que[0]);
02:  previous_value = atomic_cas(variable, Undetermined, value);
03:  if previous_value is Undetermined then
04:     variable_que[que_addr] = variable value;
05:  else
06:     variable_que[que_addr] = Canceled;
```

Fig. 5 Variable value assignment using atomic operation

균일한 개수의 0이 아닌 요소를 가지고 있으며, 각 행에서 0이 아닌 요소에 해당하는 벡터의 요소를 읽어서 그 값들로 부터 행의 값을 계산한다.

본 논문에서 제안하는 전달 루틴에서는 절들을 희소 행렬의 CSR과 같은 형태로 표현하였다. 절의 시작점들을 저장하는 배열 ptr이 있으며, 이 ptr은 문자 배열의 어느 위치에서 해당 절이 시작하는지 가리킨다. 문자 배열에서는 같은 절에 속한 문자들이 연속적으로 배치되어 있다. 문자들은 해당 변수의 첨자와 문자의 극성(positive or negative)이 결합되어서 저장된다.

이러한 절 데이터 구조를 이용한 CUDA 환경에서의 전달 루틴을 의사 코드로 표현한 것이 그림 3이다. 이 루틴을 호출할 때 절의 개수만큼 쓰레드를 생성하고 각 쓰레드들이 하나의 절에 대해 절 값을 계산하고 필요할 경우 변수 값을 지정한다. 그림 3의 루틴에서 2번 줄은 해당 쓰레드에서 처리할 절의 첨자를 결정하는 문장이고 3번~8번 줄의 반복문에서 해당 절의 문자들을 차례로 읽는다. 각 문자에 대해 해당 변수의 값과 문자의 극성을 비교하여 문자의 값이 0이면 count를 증가시키고(8번 줄) 1이면 감소시킨다(6번 줄). 반복문을 마친 뒤 count 값을 절의 길이와 비교하는데, 만일 count 값이 절 길이와 같으면 이것은 그 절의 모든 문자들의 값이 0이라는 의미이므로 충돌 상황이다(10번 줄). 그렇지 않고 만일 count 값이 절 길이 보다 1이 작다면 이것은 유닛 절임을 의미하고, 아직 값이 지정되지 않은 변수에 대해 그 값을 결정할 수 있다(12번 줄).

4.3. 아토믹(Atomic) 연산의 이용

전달 루틴 이외의 다른 루틴들은 CPU에서 진행되므로 전달 루틴의 결과, 즉 새로 전달된 변수의 값과 발견된 충돌 절에 대한 정보를 메모리에 저장해서 CPU에 전송해야 한다. 본 논문에서 제안하는 전달 루틴에서는 여러 개의 절들을 동시에 처리하므로 여러 개의 충돌을 동시에 발견할 수도 있고 여러 개의 변수들에 동시에 값을 전달할 수도 있다. 그리고 하나의 변수에 여러 쓰레드가 동시에 값을 지정하려 할 수도 있다.

이러한 상황에서 손실 없이 정보들을 메모리에 저장하기 위해 두 개의 큐를 사용한다. 첫 번째 큐는 충돌이 발생한 절을 기록하는 큐이고 두 번째 큐는 전달 과정을 통해 값이 결정된 변수를 저장하는 큐이다. 각 큐의 0번 위치는 큐 안에 저장된 데이터의 개수를 저장하도록 하였다. 각 큐에 대해 값을 저장하는 과정은 그림 4와 같이 CUDA에서 제공하는 아토믹 연산을 통해, 여러 쓰레드가 동시에 접근하더라도 문제가 없도록 하였다.

전달 과정을 통해 변수의 값을 결정할 때는 아토믹 연산을 하나 더 사용해야 한다. 여러 개의 쓰레드가 여러 절을 동시에 검사하므로 같은 변수에 대해 여러 쓰레드가 값을 지정하려 할 수도 있다. 이러한 상황에 대처하기 위해 그림 5와 같이 atomic_cas라는 루틴을 이용한다. 이 atomic_cas 루틴은 CUDA 환경에서 제공하는 아토믹 루틴 중 하나로써, 지정된 메모리 주소에 쓰기 동작을 할 때 우선 그 위치의 값을 읽어서 특정한 값과 같을 때만 쓰기 동작을 수행한다.

Table. 2 Experimental Results

SAT Problems	CUD@SAT[9] (seconds)	Proposed (seconds)	SAT Problems	CUD@SAT[9] (seconds)	Proposed (seconds)
bmc-ibm-1.cnf	139.89	1.02	par16-1-c.cnf	53.02	0.88
bmc-ibm-2.cnf	0.78	0.03	par16-1.cnf	138.87	2.87
bmc-ibm-3.cnf	> 3600	1.36	par16-2-c.cnf	251.78	5.91
bmc-ibm-4.cnf	2586.65	0.69	par16-2.cnf	260.82	2.19
bmc-ibm-5.cnf	11.33	0.08	par16-3-c.cnf	792.23	2.00
bmc-ibm-6.cnf	1598.01	1.45	par16-3.cnf	363.21	1.25
bmc-ibm-7.cnf	7.20	0.07	par16-4-c.cnf	380.24	0.42
bmc-galileo-8.cnf	> 3600	1.21	par16-4.cnf	197.04	0.18
bmc-galileo-9.cnf	> 3600	3.13	par16-5-c.cnf	388.76	1.57
bmc-ibm-10.cnf	> 3600	0.94	par16-5.cnf	72.78	2.16
bmc-ibm-11.cnf	> 3600	2.94	-	-	-
bmc-ibm-12.cnf	> 3600	19.55	-	-	-
bmc-ibm-13.cnf	> 3600	55.00	-	-	-

2번 줄에 있는 atomic_cas 루틴에서는 해당 변수의 값을 조사해서 Undetermined인지 확인하고, Undetermined가 맞는다면 그 값을 입력 받은 값으로 바꾸고, 그렇지 않으면 값을 바꾸지 않는다. 이 때 atomic_cas의 반환값은 바뀌기 전에 저장되어 있던 값이다. 3번 줄에서와 같이 그 반환값이 Undetermined라면, atomic_cas에 의해 정상적으로 변수의 값이 바뀌었을 것이므로 변수 큐에 저장한다. 그렇지 않다면 이미 다른 쓰레드에 의해 변수의 값이 지정된 상황이므로 변수 큐에는 변수 값 지정을 취소했다고 기록한다.

Ⅴ. 실험 결과

본 연구에서 제안한 알고리즘을 가장 최근에 발표된 GPGPU 기반의 SAT 알고리즘인 [9]의 알고리즘과 비교하였다. [8]과 [12]는 3-SAT에만 적용가능하고 그 프로그램이 공개되어 있지 않아서 비교에서 제외하였다. 비교한 SAT 문제들은 SATLIB[13]에 있는 문제들이다. 실험을 한 서버는 CPU가 Xeon E5-2650 2.6GHz이며 64GB 메모리를 장착하고 있다. 사용한 GPU는 NVIDIA사의 GeForce GTX 980 Ti이다.

표 2는 [9]의 알고리즘과 제안한 알고리즘을 실행한 결과이다. 표의 왼편은 SATLIB의 문제들 중 BMC(bounded model checking)[14]로부터 유도된 SAT 문제들에 대한 결과이며, 오른편은 패리티 학습 문제 [15]로부터 유도된 SAT 문제에 대한 결과이다. 각 SAT 문제에 대해 실행 시간을 측정하여 초 단위로 표에 표시하였으며, 3600초가 넘으면 실행을 중지시켰다.

표 2에 따르면 제안한 알고리즘의 성능이 [9]의 알고리즘에 비해 성능이 10배 이상 향상되었음을 알 수 있다. 논문 [9]의 알고리즘이 제한 시간 내에 해결하지 못한 문제도 해결할 수 있었다. 이것은 MiniSAT에 들어 있는 현대 SAT solver들의 이점을 활용하면서도, SpMV에서 연구된 기법들을 이용하여 GPU의 병렬 처리 능력을 활용한 결과이다.

Ⅵ. 결 론

본 연구에서는 GPGPU에 기반을 둔 SAT 알고리즘에 대해 다루었다. 전자 설계 자동화 분야에서 대표적인 알고리즘 중 하나인 SAT 알고리즘은 불규칙적인 데이터의 형태 등으로 인해 GPGPU에서 실행하기 어려운 알고리즘 중 하나이다. 본 연구에서는 기존의 단일 코어 기반의 SAT solver에서 비교적 병렬화가 용이한 전달 루틴을 GPGPU에서 실행함으로써 성능 향상을 꾀하였다. 기존의 GPGPU 기반 SAT solver와 비교하였을 때 수행시간이 단축되었음을 확인할 수 있었다. 현재의 연구에서 GPU로의 데이터 전송 시간을 줄이거나 데이터 저장 형태를 개선하는 등의 방향으로 연구를 더 진행할 수 있을 것이다.

ACKNOWLEDGMENTS

This paper was supported by the 2015 Sabbatical Year Research Program of KOREATECH.

REFERENCES

[1] Y. Deng, "GPU Accelerated VLSI Design Verification," in *Proceedings of International Conference on Computer and Information Technology,* pp. 1213-1218, 2010.

[2] Y. Deng, B. D. Wang, and S. Mu, "Taming irregular EDA applications on GPUs," in *Proceedings of International Conference on Computer Aided Design*, pp. 539-546, 2009.

[3] J. P. M.-Silva and K. A. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506-521, May 1999.

[4] M. W. Moskewicz, et al., "Chaff: Engineering an Efficient SAT Solver," in *Proceedings of Design Automation Conference*, pp. 530-535, 2001.

[5] N. Een and N. Sorensson, "An Extensible SAT-solver," in *Proceedings of International Conference on Theory and Applications of Satisfiability Testing*, pp. 502-518, 2003.

[6] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *Proceedings of International Joint Conference on Artificial Intelligence*, pp. 399-404, 2009.

[7] Y. Hamadi, S. Jabbour, and L. Sais, "ManySAT: a parallel SAT solver," *Journal on Satisfiability, Boolean Modelling and Computation*, vol. 6, pp. 245-262, Jul. 2008.

[8] H. Fujii and N. Fujimoto, "GPU acceleration of BCP procedure for SAT algorithms," in *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications,* 2012.

[9] A. D. Palu, et al., "CUD@SAT: SAT solving on GPUs," *Journal of Experimental and Theoretical Artificial Intelligence*, vol 27, no. 3, pp. 293-316, May 2015.

[10] NVIDIA, NVIDIA GeForce GTX 980 [Internet]. Available: http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_980_Whitepaper_FINAL.PDF.

[11] M. Garland, "Sparse matrix computations on manycore GPU's," in *Proceedings of Design Automation Conference,* pp. 2-6, 2008.

[12] Q. Meyer, et al., "3-SAT on CUDA: Towards a massively parallel SAT solver," in *Proceedings of International Conference on High Performance Computing and Simulation*, pp. 306-313, 2010.

[13] H. H. Hoos and T. Stutzle, "Satlib: an online resource for research on SAT," in *Proceedings of International Conference on Theory and Applications of Satisfiability Testing*, 282-292, 2000.

[14] A. Biere and et al., "Symbolic model checking using SAT procedures instead of BDDs," in *Proceedings of Design Automation Conference,* pp. 317-320, 1999.

[15] A. Blum, A. Kalai, and H. Wasserman, "Noise-tolerant learning, the parity problem, and the statistical query model," *Journal of ACM*, vol. 50, no. 4, 506-519, Jul. 2003.

강형주(Hyeong-Ju Kang)

1998년 한국과학기술원 전기및전자공학과 학사
2000년 한국과학기술원 전기및전자공학과 석사
2005년 한국과학기술원 전자전산학과 박사
2005년~2006년 (주)매그나칩반도체 선임연구원
2006년~2009년 (주)지씨티리써치 선임연구원
2009년~현재 한국기술교육대학교 컴퓨터공학부 전임강사/조교수
※관심분야 : VLSI설계 및 CAD, 마이크로프로세서 설계, 통신 모뎀 설계