AKCELERACJA OBLICZEŃ KRYPTOGRAFICZNYCH Z WYKORZYSTANIEM PROCESORÓW GPU

Patryk Bęza, Jakub Gocławski, Paweł Mral, Piotr Sapiecha*

Politechnika Warszawska,
Wydział Matematyki i Nauk Informacyjnych
*Wydział Elektroniki i Technik Informacyjnych

Streszczenie. Problem spełnialności formuł rachunku zdań *SAT* jest jednym z fundamentalnych oraz otwartych zadań we współczesnej informatyce. Jest on problemem $\mathcal{NP}$-zupełnym. To znaczy, że wszystkie problemy z klasy $\mathcal{NP}$ możemy sprowadzić do problemu *SAT* w czasie wielomianowym. Co ciekawe, wśród problemów z klasy $\mathcal{NP}$ istnieje wiele takich, które są ściśle związanych z kryptologią, na przykład: faktoryzacja liczb – ważna dla *RSA*, łamanie kluczy szyfrów symetrycznych, znajdowanie kolizji funkcji skrótu i wiele innych. Odkrycie wielomianowego algorytmu dla *SAT* skutkowałoby rozwiązaniem problemu milenijnego: $\mathcal{P}$ vs. $\mathcal{NP}$. Cel ten wydaje się bardzo trudny do osiągnięcia – nie wiadomo nawet czy jest możliwy. Mając nieco mniejsze aspiracje możemy projektować algorytmy heurystyczne lub losowe dla *SAT*. W związku z tym, głównym celem autorów pracy jest przedstawienie projektu równoległego *SAT Solvera* bazującego na algorytmie *WalkSAT*, w tym procesu jego implementacji z wykorzystaniem środowiska programistycznego *OpenCL* oraz komputera wyposażonego w karty graficzne *NVIDIA Tesla*. Wraz z dynamicznym rozwojem technologii procesorów typu *GPU* oraz układów *FPGA*, jak również przenośnością rozwiązań stworzonych w *OpenCL*, kierunek takich prac staje się interesujący ze względu na uzyskiwaną efektywność obliczeniową, jak również szybkość prototypowania rozwiązań.

Słowa kluczowe: spełnialność formuł logicznych, SAT Solver, WalkSAT, karty graficzne, kryptoanaliza, OpenCL, obliczenia równoległe.

Problem spełnialności formuł rachunku zdań (problem *SAT*) od zawsze wzbudzał zainteresowanie nie tylko matematyków, ale również programistów, którzy szukali (i dalej szukają) szybkich algorytmów, opartych na różnych heurystykach, rozwiązujących ten problem. Pojawienie się nowych technologii i ulepszanie starych, pozwala na mierzenie się z coraz większymi zadaniami problemu *SAT*. Zwiększenie taktowania procesorów jest jednym ze sposobów na wzrost liczby wykonywanych rozkazów maszynowych w jednostce czasu. Niestety, zwiększanie częstotliwości taktowania procesora wiąże się między innymi z koniecznością jeszcze większej miniaturyzacji układów scalonych, co powodowałoby problemy natury mechaniki kwantowej (efekt tunelowy) [2, 3]. Z tego powodu, zamiast zwiększać moc obliczeniową jednego procesora, dokładane są kolejne procesory, tak aby

obliczenia były prowadzone równolegle na wielu procesorach. Jednym ze sposobów na osiągnięcie masowej równoległości są multiprocesory używane w kartach graficznych.

1. Wstęp

Istnieje wiele zastosowań rozwiązań problemu spełnialności formuł rachunku zdań (*Satisfiability*, *SAT*). Jest to problem $\mathcal{NP}$-zupełny, a to znaczy że każdy z problemów z klasy $\mathcal{NP}$ (np. problem: kliki, komiwojażera, plecakowy) można zredukować do problemu *SAT* w czasie wielomianowym. W zastosowaniach kryptograficznych wiele problemów można sprowadzić do problemu *SAT* (np. faktoryzację liczb, łamanie klucza szyfru symetrycznego, kolizje funkcji skrótu). Znane są metody transformowania układów szyfrujących do postaci formuł logicznych zapisanej w szczególnej postaci, nazywanej postacią *CNF (Conjuctive Normal Form)* [13]. Dla problemów kryptograficznych, jeśli dla zadanej formuły istnieje co najmniej jedno wartościowanie, będzie interesować nas znalezienie co najmniej jednego z nich (a nie tylko odpowiedź, że takie istnieje). W praktyce takie wartościowanie może odpowiadać na przykład wartości klucza użytego w szyfrowaniu symetrycznym [1]. Znalezienie odpowiedniego wartościowania może być więc równoważne *złamaniu* szyfru, to znaczy uczynienia go niezdolnym do zapewnienia poufności szyfrowanych wiadomości.

Co roku odbywa się konkurs na najlepszy *SAT Solver* w kilku kategoriach [5]:

- SAT + UNSAT,
- SAT,
- Certified UNSAT,

w ramach których wydzielono podkategorie, w zależności od rodzaju danych wejściowych zapisanych w postaci *CNF*, czyli koniunkcyjnej postaci normalnej (koniunkcja alternatyw literałów formuły logicznej). Wyróżnia się następujące rodzaje danych wejściowych:

- Application – praktyczne problemy „z życia wzięte”,
- Hard combinatorial – trudne problemy kombinatoryczne,
- Random – problemy losowe.

Programowanie równoległe polega na wykonywaniu obliczeń w taki sposób, aby wiele instrukcji było wykonywanych jednocześnie. Taka forma programowania coraz bardziej zyskuje na znaczeniu, ze względu na zbliżanie się do granicy uniemożliwiającej dalsze zwiększanie częstotliwości taktowania

procesorów (która wiąże się z większą miniaturyzacją, która z kolei może powodować efekt tunelowy [2, 3]). Z tego względu, do przeprowadzenia obliczeń wykorzystuje się wiele procesorów równocześnie. Jednym ze sposobów uruchomienia tysięcy wątków jednocześnie jest użycie kart graficznych zgodnych z jedną z dwóch (lub obiema) technologiami przetwarzania równoległego na *GPU – OpenCL* [27] oraz *CUDA* [4]. Programowanie równoległe jest oczywiście trudniejsze w implementacji na poziomie algorytmów od programowania sekwencyjnego, gdyż współbieżność może nie tylko zwiększyć wydajność programu, ale także wprowadzić nowe miejsca, w których można popełnić błąd. Pojawiają się problemy typu: zagłodzenie, zakleszczenie, wyścig i tym podobne.

W pracy tej, zostanie przedstawiona aplikacja *GPUWalksat* realizująca równoległy *SAT Solver* (bazujący na algorytmie *WalkSAT*), zaimplementowana z wykorzystaniem środowiska programistycznego *OpenCL* oraz procesorów *GPU* (kart typu *NVIDIA Tesla*).

2. Problem spełnialności SAT

Problem spełnialności formuł rachunku zdań (*SAT*) jest problemem $\mathcal{NP}$-zupełnym. Dowód $\mathcal{NP}$-zupełności problemu *SAT* został opublikowany w roku 1971, przez Amerykanina, profesora Stephena Arthura Cooka, za co między innymi, 11 lat później, został nagrodzony Nagrodą Turinga [14].

Sformułowanie problemu:
Wejście: Poprawnie zapisana formuła rachunku zdań F o n zmiennych.
Wyjście: Czy istnieje $\vec{X} \in \{0,1\}^n$ taki, że $F(\vec{X}) = 1$?

Przykład: Dla formuły rachunku zdań:

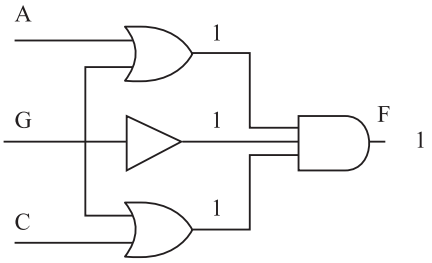
$$F(A,B,C) = (A \vee B) \wedge (B \vee C) \wedge \neg B,$$

którą można w następujący sposób przedstawić za pomocą bramek logicznych: odpowiedzią na tak postawione pytanie jest **TAK**. Jedynym rozwiązaniem jest wektor:

$$\vec{X} = (A,B,C) = (1,0,1)$$

taki, że:

$$F(1,0,1) = (1 \vee 0) \wedge (0 \vee 1) \wedge \neg 0 = 1$$



3. Zastowania w kryptografii

Jednym z głównych zastosowań problemu *SAT* w kryptografii są ataki algebraiczne na szyfry strumieniowe – *Keeloq* oraz blokowe – *DES*. Ideą ataków algebraicznych jest przekształcenie ataku na system kryptograficzny w rozwiązanie układu równań wielomianowych. Głównym problemem tego podejścia jest fakt, że rozwiązanie układu równań wielomianowych nad skończonym ciałem jest problemem $\mathcal{NP}$-zupełnym [26]. Jednak redukcja tego problemu do problemu *SAT* daje możliwość uzyskania skutecznych ataków.

4. Algorytmy stosowane w SAT Solverach

W związku z dużą złożonością algorytmu brutalnego rozwiązującego problem *SAT*, który wymagałby sprawdzenia wszystkich możliwych 2^n wartościowań, trwają poszukiwania algorytmów o lepszych złożonościach. Opracowanie algorytmu o złożoności wielomianowej byłoby równoważne z udowodnieniem, że $\mathcal{P} = \mathcal{NP}$, a zatem rozwiązaniem jednego z siedmiu problemów milenijnych ogłoszonych w 2000 roku przez Instytut Matematyczny Claya [19]. Powstało wiele algorytmów skracających czas potrzebny do otrzymania wyniku. Najpopularniejszym z nich jest algorytm *DPLL (DavisPutnamLogemannLoveland algorithm)* używany w odnoszących sukcesy *SAT Solverach* takich jak *MiniSat* [6], *Chaff* [7] czy *Glucose* [8].

Algorytm ten, jest algorytmem z powrotami (*backtracking*) działającym poprzez wybór literału, przypisanie mu wartości logicznej, uproszczenie formuły, a następnie rekursywne sprawdzenie czy uproszczona formuła jest spełnialna. Jeśli kolejne wywołania zwrócą odpowiedź twierdzącą oznacza to, że początkowa formuła jest spełnialna, w przeciwnym przypadku podobne rekursywne sprawdzenie jest wykonywane przy założeniu przeciwnej wartości logicznej. Krok uproszczenia przede wszystkim polega na usunięciu klauzul, które stają się prawdziwe przy obecnym wartościowaniu formuły oraz tych literałów znajdujących się w pozostałych klauzulach,

które stały się fałszywe. Mimo to, pesymistyczna złożoność tego algorytmu jest jednak nadal wykładnicza

Obecnie opracowywane są struktury danych oraz heurystyki pozwalające usprawnić działanie tego typu *SAT Solverów*. Znaczny wpływ na poprawę skuteczności ma też zastosowanie strategii pozwalających na uczenie się algorytmu oraz nawracanie o więcej niż jeden węzeł w przypadku wykrycia konfliktu *backjumping* [13].

Kolejnym podejściem do rozwiązania problemu *SAT* są algorytmy przeszukiwania lokalnego, wśród których szczególną wydajnością wyróżniają się algorytm *Walksat* [9] oraz rodzina algorytmów *Novelty* [10]. Opierają się one danym początkowym wartościowaniu zmiennych formuły logicznej, dla którego przy pomocy różnych heurystyk wybierane są zmieniane, których wartości są następnie zmieniane na przeciwne. W ten sposób algorytm minimalizują liczbę niespełnionych klauzul zbliżając się tym samym do rozwiązania. Dodatkowo wprowadzana jest pewna losowość celem uniknięcia utknięcia w lokalnym minimum.

5. SAT Solvery równoległe

Podjęto także próby zrównoleglenia *SAT Solverów*. W tej dziedzinie warty wspomnienia jest *GrADSAT* [15] stworzony na Uniwersytecie Kalifornijskim w Santa Barbara, który wykorzystuje *solver zChaff* oraz architekturę typu *mater-slave* zrealizowaną na siatce (*grid*). Problem dzielony jest w nim na podprzestrzenie i rozwiązywany przez niezależnych klientów dzielących między sobą zbiór nauczonych klauzul. Podobnie zrealizowano także *gNovelty+ (v.2)* z tą różnicą, że tu każdy klient używa algorytmu gNovelty+ do rozwiązania problemu [16].

Inne podejście zastosowano w *SAT Solverze ManySAT* [17], stworzonym przez Microsoft Research przy współpracy z Universite dArtois, w którym to uruchamianych jest równolegle wiele instancji algorytmu *DPLL*. Jednak każda z instancji używa innych heurystyk, strategii restartów oraz sposobów uczenia. Strategia ta dała bardzo dobre efekty i w 2009r. *ManySAT* został zwycięzcą konkursu *SAT*.

W *GPU4SAT* powstałym na Universite de Reims Champagne-Ardenne autorzy proponują sprowadzenie problemu *SAT* do mnożenia macierzy i wykorzystanie procesorów graficznych do zrównoleglenia tej operacji [11]. Wykorzystywane jest tu podejście niezupełne, a kolejne wartościowania wybierane są za pomocą heurystyk.

Z kolei *PGSAT* zaprojektowany na Uniwersytecie w Bolonii jest zrównoleglonąą wersją algorytmu *GSAT (algorytm zachłanny)* [18]. Każdy wątek otrzymuje własne wartościowanie zmiennych, a następnie przy pomocy

funkcji oceny wybierana jest najlepsza zmienna do zamiany. Proces powtarzany jest aż do otrzymania wartościowania spełniającego zadaną formułę.

6. Procesory GPU

Początek historii procesorów graficznych, w skrócie *GPU*, sięga historii pierwszych komputerów, których wynik działania pojawiał się na ekranie. Pierwsze karty graficzne miały za zadanie jedynie wyświetlać znaki alfanumeryczne. Z czasem możliwe stało się użycie ekranów, na których można było wyświetlać i zmieniać kolory pojedynczych pikseli. Wyświetlanie wyników działania komputera przestało ograniczać się do wyświetlania symboli alfanumerycznych i poszerzyło się o możliwość wyświetlania obrazów, wykresów, diagramów i innych, graficznych, sposobów reprezentacji danych.

Dziś karty graficzne weszły na kolejny etap rozwoju, w którym zaczęto wykorzystywać ich możliwości obliczeniowe do wsparcia obliczeń dotychczas wykonywanych jedynie na procesorze głównym (*CPU*). Producenci niektórych kart graficznych rozszerzyli możliwości wykorzystania *GPU* do równoległego przetwarzania danych, udostępniając własny język programowania oraz interfejs programistyczny (*API*), umożliwiający programowanie równoległego przetwarzania danych, niekoniecznie związanych z grafiką komputerową.

Karty graficzne zostały docenione przez twórców jednego z najszybszych komputerów znajdujących się na Ziemi. Amerykański superkomputer *Titan* o mocy obliczeniowej 17,59 PFLOPS, znajdujący się w *Oak Ridge National Laboratory*, który od listopada 2012 do czerwca 2013, był najszybszym superkomputerem na świecie[1], wykorzystuje do obliczeń, poza 18688 procesorami *AMD Opteron 6274 CPU*, taką samą liczbę kart graficznych *NVIDIA Tesla K20X*[2] [20, 21]. Pracownicy instytutu w *Oak Ridge*, przed otrzymaniem dostępu do potężnej mocy obliczeniowej superkomputera, przechodzą szkolenie z programowania kart graficznych, tak aby jak najwydajniej wykorzystywać każdy cykl zegara multiprocesora karty graficznej. [23]

[1] w czasie pisania tego artykułu jest plasowany na drugiej pozycji w rankingu najszybszych superkomputerów, za chińskim *Tianhe-2*

[2] w czasie pisania tego artykułu cena za sztukę to około 3000$ [22]

7. OpenCL vs CUDA

Dwie, obecnie najpopularniejsze, technologie wspierające równoległe programowanie przy użyciu procesorów graficznych to *OpenCL* i *CUDA* [4, 27]. Pierwsza z nich jest wieloplatformowym, otwartym standardem, to znaczy zastosowanie *OpenCL* nie ogranicza się do jednej platformy sprzętowej, jak w przypadku *CUDA*, dedykowanej jedynie dla sprzętu amerykańskiej firmy *NVIDIA* (która również angażuje się w projekt *OpenCL*). Używając technologii *CUDA* jesteśmy ograniczeni do programowania sprzętu firmy *NVIDIA*, natomiast *OpenCL* działa na wielu innych środowiskach (takich jak *FPGA*, *CPU*, *DSP*), obsługując również część urządzeń *NVIDII*.

Pisząc kod zarówno w *CUDA*, jak i *OpenCL*, programista pisze z perspektywy wykonania pojedynczego wątku. Pojęcie wątku może być mylnie rozumiane przez programistów, którzy programowali wątki jedynie na *CPU* (na przykład korzystając z popularnej implementacji POSIX wątków o nazwie *pthreads*), ponieważ wątki na *GPU* charakteryzują się tym, że wykonują się jednocześnie, pod warunkiem, że realizują dokładnie ten sam kod programu dla (potencjalnie) różnych danych. Dla przykładu: jeśli użyjemy instrukcji `if-else`, tak, że w i-tym wątku wykonuje się warunkowo instrukcja A_i, a w wątku $j \neq i$ instrukcja $A_j \neq A_i$, to taki program nie będzie wykonywał się równolegle na *GPU*. Jest to ograniczenie względem wątków znanych z *CPU*, jednak dzięki niemu jest możliwe uruchomienie znacznie większej liczby wątków[3] niż na zwykłym procesorze komputera. Przedstawiony sposób działania programu jest nazwany działaniem *SIMD*, tzn. *Single Instruction, Multiple Data* i jest jedną z kategorii wyróżnionych w *Taksonomii Flynna*.

CUDA

Aby najlepiej oddać czym jest *CUDA* warto odwołać się do twórców tej technologii, czyli do firmy *NVIDIA*, która na swojej stronie internetowej opisuje technologię *CUDA* następująco: "*CUDA* to opracowana przez firmę *NVIDIA* równoległa architektura obliczeniowa, która zapewnia radykalny wzrost wydajności obliczeń, dzięki wykorzystaniu mocy układów GPU (*Graphics Processing Unit* - jednostka przetwarzania graficznego) [4]. Określenie wzrostu wydajności *radykalnym* wydaje się uzasadnione w świetle przykładowych aplikacji udostępnianych przez firmę *NVIDIA*, napisanych zarówno w wersji na *CPU*, jak i *GPU* w celu porównania wydajności

[3] rzędu tysięcy

[28]. Przyspieszeni a niektórych wersji aplikacji na *GPU* jest rzędu kilkudziesięciu razy względem tych na *CPU*.

Firma *NVIDIA*, w ramach wsparcia technologii *CUDA*, dostarcza środowisko programistyczne, składające się m.in. z: kompilatora *nvcc*, debuggera *cuda-gdb* oraz *profilera*, to znaczy aplikacji umożliwiającej pomiar wydajności (na przykład poprzez śledzenie czasu zajętości multiprocesora karty graficznej).

Istnieje szeroki wybór kart graficznych firmy *NVIDIA* opartych o różne mikroarchitektury. Ich nazwy, w kolejności ich powstania to: *Tesla*, *Fermi*, *Kepler*, *Maxwell* i *Pascal*. Chcąc programować w technologii *CUDA* warto mieć sprzęt z możliwie najwyższym numerem *compute capability*, ponieważ karty graficzne z wyższym *compute capability* są wzbogacone o nowe funkcje i mechanizmy *CUDA*[4].

Jedną z najnowszych i najdroższych kart graficznych *NVIDII* jest *NVIDIA Tesla K40* i kosztuje około 5000$ za sztukę [24]. Istnieją projekty, które mają wykorzystywać moc obliczeniową tej karty. W *Texas Advanced Computing Center* w Austin, w stanie Teksas, ma powstać centrum o nazwie *Maverick*, służące do zdalnej wizualizacji i analizy danych (*Remote Visualization and Data Analysis*) z wykorzystaniem między innymi 132 kart *Tesla K40* [25].

OpenCL

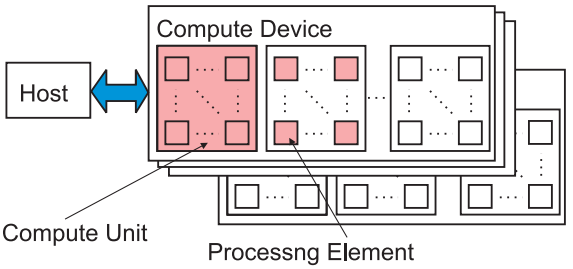
Idea działania *OpenCL* jest bardzo zbliżona do architektury *CUDA*. Dynamiczny sposób kompilacji kodu wykonywanego na maszynie (to jest po uruchomieniu programu), idea działania wątków podzielonych na bloki i wiele innych mechanizmów jest takich jak w *CUDA*. Różnice między tymi technologiami polegają na innych interfejsach programistycznych (*API*), to znaczy w funkcjach obsługujących sprzęt. *OpenCL* wydaje się być nieco bardziej niskopoziomowym podejściem w kodowaniu niż *CUDA*. Ponadto istotną różnicą względem *CUDA* jest otwartość *OpenCL*. Zarówno pod względem otwartości kodu całego projektu, jak i otwartości na inne urządzenia, nie tylko karty graficzne. *OpenCL* potrafi obsługiwać wiele urządzeń jednocześnie (tak jak *CUDA*), jednak urządzenia mogą być urządzeniami różnych typów (nie tylko *GPU*). Co więcej, dzięki aplikacji *VirtualCL* możemy łatwo zrównoleglić wykonanie programów napisanych w *OpenCL*, na wiele komputerów połączonych w sieć [29].

[4] na przykład przed powstaniem architektury o nazwie *Fermi* nie było możliwe korzystanie z rekurencji

Aby zrozumieć szczegółowy, niskopoziomowy sposób działania architektury *OpenCL* warto przeanalizować jak *OpenCL* korzysta ze sprzętu, na którym są wykonywane programy napisane w języku *OpenCL C*. Niestety, w ogólności, szczegółowa analiza wykonania kodu napisanego na platformy wspierające *OpenCL* jest utrudniona z powodu szerokiej gamy urządzeń wspierających *OpenCL*. Każda architektura sprzętowa wspierana przez *OpenCL* ma inną budowę i specyfikę działania, dlatego do opisu działania *OpenCL* wygodnie jest użyć pojęć będących abstrakcyjnym ujęciem faktycznej realizacji sprzętowej (można to nazwać enkapsulacją/hermetyzacją sprzętowego wykonania aplikacji). Szczegółowa realizacja wykonania programu (to jest na przykład obsługa przydziału wątków do poszczególnych procesorów sprzętowych, kolejność alokowania pamięci i tak dalej) jest zależna od konkretnego sprzętu i implementacji standardu *OpenCL* przez producenta sprzętu na którym wykonywany jest program. W celu przeprowadzenia dokładnej analizy sprzętowej wykonania programu, należy zapoznać się ze specyfikacją producenta sprzętu, na którym jest on uruchamiany (o ile jest udostępniona) [30].

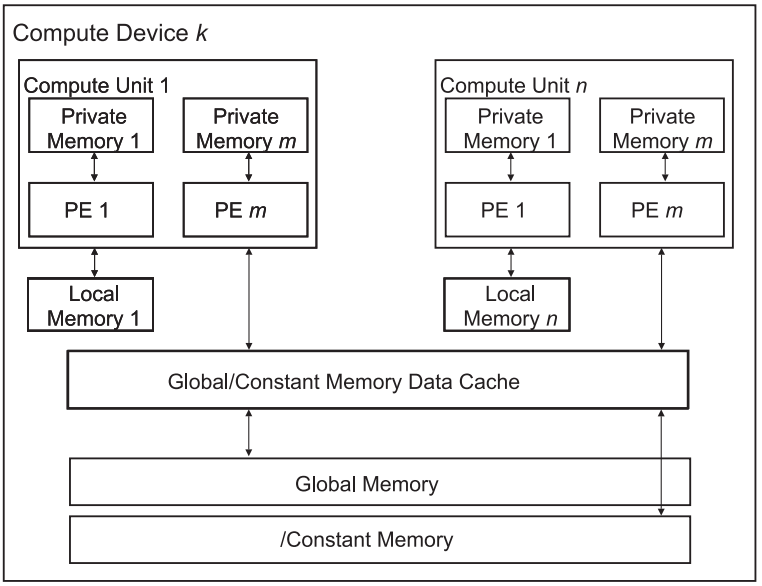
Prosty schemat bloków złożonych z wątków programu w architekturze *OpenCL* przedstawia poniższy

Jeśli program wykonuje się na karcie graficznej, to wątki wykonują się w blokach na jednostkach obliczeniowych (*compute units*), będących częścią multiprocesora karty graficznej.



OpenCL ma do dyspozycji kilka rodzajów pamięci (w kolejności rosnącej szybkości dostępu):

- pamięć globalną – rzędu gigabajtów,
- pamięć stałych – to jest wydzieloną częścią globalną z optymalizacjami, z racji niezmienności danych,
- pamięć lokalna – współdzielona między wątkami w jednym wątku, rzędu kilkudziesięciu kilobajtów,
- pamięć prywatna – niewspółdzielona pamięć dla jednego wątku.



8. Implementacja

Proponowane przez nas rozwiązanie, czyli implementacja *SAT Solvera* na karcie graficznej (*GPU*), korzysta z algorytmu *WalkSAT*, a uruchamiane jest na 3 kartach *NVIDIA Tesla M2070* dostępnych w laboratorium *Wydziału MiNI PW*.

Ogólne założenia implementacji sprowadzają się do tego, by uruchomić jak najwięcej instancji algorytmu *WalkSAT* na jak największej liczbie wątków na karcie graficznej (dokładniej: na każdej z trzech dostępnych kart graficznych). Każda instancja algorytmu otrzymuje inne wartościowanie początkowe, które jest dla niej punktem startowym obliczeń. Jeśli którakolwiek instancja algorytmu znajdzie wartościowanie spełniające zadaną formułę logiczną, obliczenia są przerywane, a wyniki są zapisywane do pliku.

Im mniejszy problem, tym więcej instancji algorytmu można uruchomić i tym szybciej zostanie rozwiązany.

9. Architektura rozwiązania

Program dzieli są na dwa główne moduły: moduł uruchamiany na procesorze głównym komputera (*CPU*) oraz na moduł uruchamiany na karcie graficznej (*GPU*).

Moduł CPU

Program *GPUWalksat* rozpoczyna wykonanie od przygotowania danych dla karty graficznej, które odbywa się na *CPU*. W tym module wyczytywana jest formuła logiczna *CNF* zapisana w formacie *DIMACS*. Na jej postawie, przygotowywane są struktury danych dla *GPU* oraz konfigurowane jest środowisko *OpenCL*. W celu wykorzystania wszystkich 3 dostępnych kart *NVIDIA Tesla*, uruchamiane są 3 niezależne wątki *POSIX-owe* na *CPU*, tak aby każdy miał przypisaną do siebie jedną z kart graficznych. Każdy z wątków przeprowadza wspominaną konfigurację dla przypisanej do siebie karty graficznej.

Struktury danych

W ramach przygotowania środowiska, tworzone są struktury danych, które następnie przekazywane są do środowiska *OpenCL* (moduł GPU). W ich skład wchodzą:

- klauzule tworzące formułę logiczną (potrzebne do sprawdzania spełnialności),
- mapowanie zmienna-¿klauzula (potrzebne do szybkiego znajdowania klauzul zawierających daną zmienną logiczną),
- wartościowania dla każdego wątku na karcie graficznej,
- inne struktury pomocnicze potrzebne do realizacji powyższych struktur.

Warto zaznaczyć, że wszystkie struktury oprócz wartościowań, są wspólne z perspektywy wątków na *GPU*. Jedynie wartościowania są przydzielane dla każdego wątku osobno - dla każdego wątku losowane jest inne wartościowanie początkowe, które przekazywane jest w osobnej strukturze danych.

W czasie konfigurowania środowiska *OpenCL* dobierana jest także optymalna liczba wątków do uruchomienia na jednej karcie graficznej. „Wąskim gardłem" obliczeń na *GPU* jest dostępna pamięć operacyjna, której przy większych problemach zaczyna szybko brakować. W tym celu należy optymalizować struktury danych oraz liczbę wątków do uruchomienia na *GPU*. Ich liczba zależy od wielkości problemu według zasady: im większy problem, tym więcej pamięci operacyjnej na niego potrzeba, a w konsekwencji – tym mniej wątków na *GPU* można uruchomić.

Po wykonaniu wszystkich powyższych czynności, mając już skonfigurowane środowisko *OpenCL* pod kątem danego problemu do rozwiązania,

kompilowany i uruchamiany jest *kernel*, czyli kod wykonywany na karcie graficznej.

Po zakończeniu obliczeń przez moduł GPU, wykonywanie programu wraca na *CPU*, gdzie odczytywane są znalezione na *GPU* wartościowania oraz weryfikowana jest ich poprawność.

Moduł GPU

W środowisku *OpenCL* kod wykonywany na urządzeniu nazywany jest *kernelem*. Implementuje się go w języku zbliżonym do klasycznego *C* z kilkoma rozszerzeniami specyficznymi dla *OpenCL*. Każdą instrukcję rozważa się z perspektywy pojedynczego wątku (w sensie *GPU*, a nie *CPU*).

Wątek na karcie graficznej otrzymuje dane przygotowane dla niego wcześniej przez moduł *CPU*, opisane w sekcji Struktury danych. Następnie, korzystając z wylosowanego dla niego wartościowania początkowego, uruchamia pętlę algorytmu *WalkSAT*. Wartościowanie jest modyfikowane zgodnie z heurystyką algorytmu aż do momentu znalezienia rozwiązania przez dany wątek lub którykolwiek inny.

Synchronizacja

Wątki na *GPU* są od siebie w znacznej części całkowicie niezależne. Korzystają z niektórych współdzielonych struktury danych, przechowywanych w pamięci globalnej karty graficznej, jednak dostęp do niej jest automatycznie synchronizowany i kontrolowany przez kartę graficzną. Ponadto, jeśli jakiś wątek znajdzie wartościowanie spełniające zadaną formułę logiczną, ustawia flagę w pamięci globalnej, aby poinformować inne wątki, że powinny skończyć obliczenia. Flaga ta jest sprawdzana co ustaloną liczbę obrotów pętli algorytmu, tak aby niepotrzebnie nie spowalniać obliczeń.

10. Testy

Aplikacja *GPUWalksat* została poddana trzem rodzajom testów:

- testom funkcjonalnym, których zadaniem było sprawdzenie, czy zostanie znalezione rozwiązanie problemu, który wiadomo, że jest spełnialny,
- testom wydajnościowym, których celem było zmierzenie czasu rozwiązywania problemów,
- testom porównawczym, których celem było porównanie aplikacji z innymi *SAT Solverami*.

Środowisko testowe

Aplikacja *GPUWalksat* była uruchamiana na klastrze obliczeniowym dostępnym na *Wydziale MiNI PW*, złożonym z 3 kart *NVIDIA Tesla M2070*. W trakcie testów porównawczych, *SAT Solvery* działające na *CPU* uruchamiane były na komputerze osobistym z procesorem *Intel Core i7-3770* o taktowaniu 3.40 Ghz.

Wyniki testów

Testy funkcjonalne polegały na rozwiązaniu formuły złożonej z praw de Morgana o różnych stopniu złożoności. Rozwiązania zostały znalezione poprawnie. Ponadto przetestowano działanie programu na formule, o której wiadomo, że nie posiada rozwiązania. Zgodnie z oczekiwaniami, program nie znalazł rozwiązania w określonym z góry czasie.
Testy wydajnościowe i porównawcze polegały na znalezieniu rozwiązania problemów różnych typów. Zmierzony czas porównany został z czasem znajdowania rozwiązania dla tych samych problemów przez dwa inne *SAT Solvery*:

- *WalkSAT CPU* – implementacja algorytmu *WalkSAT*, jednowątkowa, na *CPU*,
- *miniSAT* – jeden z najlepszych i najbardziej znanych *SAT Solverów*, oparty na algorytmie *DPLL*, działający na *CPU*.

11. Wnioski

Testy funkcjonalne, wydajnościowe i porównawcze przeprowadzone na wielu przykładach pochodzących z *SAT Competition* [5] wykazały, że aplikacja *GPUWalksat* łącząca algorytm przeszukiwania lokalnego z wykorzystaniem kart graficznych pozwala znacznie przyspieszyć obliczenia dla pewnej grupy problemów. Na szczególną uwagę zasługują problemy losowe, a wśród nich *5-SAT* o rozkładzie jednostajnym o 120 zmiennych i 2556 klauzulach, dla którego po uruchomieniu na klastrze obliczeniowym składającym się z trzech kart *NVIDIA Tesla M2070*, rozwiązanie zostało znalezione w 68,195 s, podczas gdy obliczenia *CPUWalksata* oraz *Minisata* musiały zostać przerwane ze względu na maksymalny zakładany czas obliczeń, tj. 15 minut. Większość problemów nadal efektywniej rozwiązywana jest przez *SAT Solvery* oparte na algorytmie *DPLL*. W grupie tej znajduje się między innymi problem znajdywania rozwiązania dla quasi-grupy

Tabela 1

Wyniki zbiorcze testów wydajnościowych i porównawczych

n — liczba zmiennych, m — liczba klauzul					
Nazwa pliku z CNF	**n**	**m**	**WalkSAT CPU**	**GPUWalksat**	**miniSAT**
f100	100	400	0,02 s	0,099 s	0,004 s
f300	300	1200	0,03 s	1,313 s	0,028 s
f500	500	2150	0,57 s	11,104 s	$>$ 5 min
2000	2000	8000	11,48 s	99,567s	$>$ 5 min
f10k	10000	40000	7,45 s	$>$ 5 min	$>$ 5 min
fact_25_52	52	187	0,008 s	0,046 s	0,004 s
hard_100	100	800	0,008 s	0,534 s	0,004 s
3colorflat_200_600	600	2237	0,090 s	7,877 s	0,012 s
f16b_56893_658v_2563c	658	2563	342,59 s	33,512 s	0,008 s
5colormorph_100_500	500	3100	$>$ 5 min	25,951 s	0,004 s
quasigroup_343	343	68083	$>$ 5 min	32,689 s	0,024 s
sgen1-sat-140-100	140	336	$>$ 15 min	0,635 s	10,368 s
unif-k5-r21.3	120	2556	$>$ 15 min	68,195 s	$>$ 15 min

mający 343 zmienne i 68083 klauzul, dla którego to *GPUWalksat* znalazł rozwiązanie po 32,689 s, podczas gdy *Minisat* zakończył obliczenia w zaledwie 0,024 s.

Przedstawiona aplikacja może być zoptymalizowana na wiele sposobów w celu osiągnięcia lepszych efektów. Po pierwsze poprzez użycie modelu *SIMD* w bardziej dokładny sposób oraz poprzez optymalne wykorzystanie różnych typów pamięci dostępnych w architekturze *GPU* w celu zmniejszenia kosztów dostępu do pamięci. Poprawę wydajności można także uzyskać skupiając się na wybranym problemie, poprzez stworzenie struktur danych oraz preprocesora odpowiadających specyfice danego zagadnienia. W tym przypadku parametry algorytmu *Walksat* również powinny zostać przeanalizowane i odpowiednio dobrane.

Użycie środowiska *OpenCL* pozwala także na przeniesienie zaprezentowanego rozwiązania na układy *FPGA*, które zapewniają wysoką efektywność.

Literatura

[1] T. Eibach, E. Pilz, and G. Völkel, *Attacking Bivium Using SAT Solvers*, University of Ulm, Institute of Theoretical Computer Science
http://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.190 /Mitarbeiter/eibach/Attacking-Bivium-Using-SAT-Solvers.pdf, 01.02.2014.

[2] M. Luisier, A. Schenk, *IEEE TRANSACTIONS ON ELECTRON DEVICES*,
http://www.iis.ee.ethz.ch/ schenk/04495122.pdf, 24.04.2014.

[3] S. Anthony, 7nm, 5nm, 3nm: The new materials and transistors that will take us to the limits of Moores law,
http://www.extremetech.com/computing/162376-7nm-5nm-3nm-the--new-materials-and-transistors-that-will-take-us-to-the-limits-of-moores-law, 24.04.2014.

[4] NVIDIA Website,
http://www.nvidia.pl/object/cuda-parallel-computing-pl.html, 24.04.2014.

[5] SAT Competition, *Strona internetowa konkursu SAT Competition*, http://www.satcompetition.org/, 01.05.2014.

[6] N. Een, N. Sörensson, *An Extensible SAT-solver*, [W:] *SAT*, pod red. Enrico Giunchiglia, Armando Tacchella, Springer, 2003, s. 502–518.

[7] C. F. Madigan, S. Malik, M. W. Moskewicz, L. Zhang, Y. Zhao, *Chaff: Engineering an Efficient SAT Solver*, 39th Design Automation Conference (DAC 2001), ACM, 2001.

[8] Glucose, *The Glucose SAT Solver*.
http://www.labri.fr/perso/lsimon/glucose/, 17.04.2014.

[9] WalkSAT, *WalkSAT Home Page*,
http://www.cs.rochester.edu/u/kautz/walksat/, 17.04.2014.

[10] H. H. Hoos, D. A. D. Tompkins, *Novelty+ and Adaptive Novelty+*, https://cs.uwaterloo.ca/ dtompkin/papers/satcomp04-novp.pdf, University of British Columbia, 2004.

[11] H. Deleau, Ch. Jaillet, M. Krajecki, *GPU4SAT: solving the SAT problem on GPU*,
CReSTIC SysCom, Universitae de Reims Champagne-Ardenne, 2008.

[12] V. W. Marek, *Introduction to Mathematics of Satisfiability*, CRC Press, 2008.

[13] G. V. Bard, *Algorithms for Solving Linear and Polynomial Systems of Equations over Finite Fields with Applications to Cryptanalysis*, Rozprawa doktorska, 2007.

[14] S. Cook, *The complexity of theorem proving procedures*, Proceedings of the Third Annual ACM Symposium on Theory of Computing, 1971, s. 151–158.
[15] W. Chrabakh, R. Wolski, *GrADSAT: A Parallel SAT Solver for the Grid*, University of California Santa Barbara, 2003.
[16] D.N. Pham, C. Gretton, *gNovelty+ (v.2)* [W:] *Solver description*, SAT competition, 2009.
[17] Y. Hamadi, S. Jabbour, L. Sais, *ManySAT: A Parallel SAT Solver*, Journal on Satisfiability, Boolean Modeling and Computation, JSAT 6, 2009, s. 245–262
[18] A. Roli, *Criticality and Parallelism in Structured SAT Instances*, [W:] *CP'02. Volume 2470 of LNCS.*, pod red. P. V. Hentenryck, 2002, s.714–719.
[19] Instytut Matematyczny Claya http://www.claymath.org/millennium, 07.05.2014.
[20] top500.org, Titan – Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x, http://www.top500.org/system/177975, 05.02.2014.
[21] top500.org, Oak Ridge Claims No. 1 Position on Latest TOP500 List with Titan, http://www.top500.org/blog/lists/2012/11/press-release/, 05.02.2014.
[22] amazon.com, http://www.amazon.com/nVidia-K20X-NVIDIA-Tesla/dp/B00CX35KU2, 28.04.2014.
[23] Oak Ridge Computing Facility, https://www.olcf.ornl.gov/support/, 24.04.2014.
[24] computer store – newegg.com, http://www.newegg.com/Product/Product.aspx?Item=N82E16814132015, 24.04.2014.
[25] Texas Advanced Computing Center, https://www.tacc.utexas.edu/news/press-releases/2013/tacc-to-deploy-maverick, 28.04.2014.
[26] M. Vörös, *Algebraic attack on stream ciphers*, praca magisterska, Bratislava, 2007.
[27] Khronos Group – OpenCL, http://www.khronos.org/opencl/, 28.04.2014.
[28] CUDA Examples, http://www.nvidia.pl/object/gpu-computing-applications-pl.html, 28.04.2014.
[29] VirtualCL, http://www.mosix.org/txt_vcl.html/, 28.04.2014.

[30] Timo Stich, *OpenCL on NVIDIA GPUs*, http://sa09.idav.ucdavis.edu/docs/SA09_NVIDIA_IHV_talk.pdf, 09.02.2014.

ACCELERATION OF CRYPTOGRAPHIC CALCULATIONS USING GPUS

Abstract. The boolean satisfiability problem *SAT* is one of the fundamental and open tasks in present-day information science. This problem is $\mathcal{NP}$-complete. It means that all $\mathcal{NP}$ problems can be reduced to *SAT* in polynomial time. Interestingly, among $\mathcal{NP}$ problems, there are many closely related to cryptology, for example: factorization of numbers – important for *RSA*, breaking keys of symmetric ciphers, finding collisions of hash functions and many others. The discovery of the polynomial algorithm for *SAT* would result in resolving one of Millennium Prize Problems: $\mathcal{P}$ vs. $\mathcal{NP}$. This objective seems to be hard to achieve – it's unknown if it is even possible. With slightly lower aspirations, we can design heuristic or random algorithms for *SAT*. Therefore, the main goal of our study is to present a project of parallel *SAT Solver* based on *WalkSAT* algorithm, including its implementation using the *OpenCL* programming environment and a computer equipped with *NVIDIA Tesla* graphics cards. With the rapid development of *GPU* technology and *FPGAs*, as well as portability of solutions created in *OpenCL*, the direction of such works becomes interesting because of computational efficiency gained, as well as solution prototyping rate.

Keywords: satisfiability, SAT Solver, WalkSAT, graphics processing unit, cryptoanalysis, OpenCL, parallel computing.