	Politechnika Krakowska im. Tadeusza Kościuszki Wydział Fizyki, Matematyki i Informatyki	
Programowanie równoległe i rozproszone		
Projekt: *Opracowanie programu rozwiązującego SUDOKU za pomocą technologii CUDA.*		
Data oddania: 17-06-2018	**Kierunek:** *Informatyka Stosowana* **Rok akademicki:** *2017/2018*	**Wykonał:** *Przemysław Kleszcz,* *Krzysztof Tatar* **Nr albumu:** *124624* *127984*

Spis treści

1 Wprowadzenie **3**

2 Cel projektu **3**

3 Środowisko implementacyjne **3**

4 Czym jest SUDOKU **4**

5 Wybór algorytmu **4**

6 Implementacja **5**

7 Wnioski **6**

1 Wprowadzenie

W zakresie projektu zaliczeniowego zrealizowano projekt udowadniający możliwość wykorzystania oprogramowania CUDA w celu zbudowania aplikacji rozwiązującej diagramy Sudoku. CUDA to opracowana przez firmę Nvidia uniwersalna architektura procesorów wielordzeniowych, umożliwiająca wykorzystanie ich mocy obliczeniowej do rozwiązywania ogólnych problemów numerycznych w sposób wydajniejszy niż w tradycyjnych, sekwencyjnych procesorach ogólnego przeznaczenia. Powyższa platforma w doskonały sposób nadaje się do zastosowań takich jak Sudoku – gdzie możliwość wykonania setek obliczeń jednocześnie, w znaczący sposób zwiększa wydajność programu.

2 Cel projektu

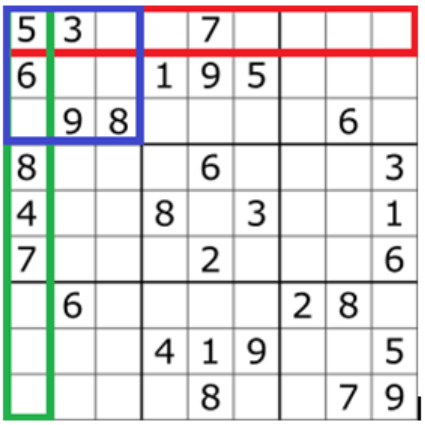
Celem projektu jest zbudowanie aplikacji, która byłaby w stanie obliczyć diagram Sudoku o jak największym stopniu skomplikowania. Wymaga to przygotowania odpowiedniej liczy diagramów początkowych – z pustymi lukami oznaczającymi możliwość wstawienia liczby, doboru optymalnego algorytmu, który byłby w stanie dokonać obliczeń w satysfakcjonującym czasie oraz, którego budowa predysponuje do optymalnego zrównoleglenia oraz środowiska, które wspomoże odpowiednio proces implementacji i debugowania.

3 Środowisko implementacyjne

Implementacja została w pełni opracowana w środowisku Microsoft Visual Studio. Jest to zintegrowane środowisko produkcyjne (IDE) zaprojektowane przez firmę Microsoft. Jest używane do produkcji programów komputerowych na platformę Windows. Za jego pomocą można produkować zarówno kod natywny jak i zarządzany przez platformę .NET. Oprócz pełnego wsparcia dla platformy .NET, Visual Studio oferuje wiele narzędzi i dodatków usprawniających proces implementacji dla wielu innych technologii – również dla języka C++ i technologii CUDA.

4 Czym jest SUDOKU

Sudoku jest łamigłówką, której celem jest wypełnienie diagramu 9 x 9 w taki sposób, aby w każdym wierszu, w każdej kolumnie i w każdej z dziewięciu podtablic 3 x 3 zwanych często blokami – znalazło się po jednej unikalnej cyfrze w zakresie od 1 do 9. Zasady przypominają bardzo tzw kwadrat łaciński, wymyślony i badany przez średniowiecznych matematyków. W Sudoku w przeciwieństwie do kwadratu łacińskiego, cyfry nie mogą się powtarzać nie tylko w każdym wierszu i kolumnie ale także w mniejszych podsekcjach 3 x 3.



Sudoku z widocznymi miejscami do uzupełnienia i zaznaczonymi obszarami, w których wymagana jest unikalność liczby.

5 Wybór algorytmu

Rozważając wybór algorytmu, który byłby w stanie obliczyć diagramy o różnych stopniach trudności oraz dał się w mało kłoptliwy sposób zrównoleglić, naturalnym kandydatem była metoda Brute Force. Standardowym podejściem wykorzystania tej metody w Sudoku jest wyprodukowanie wszystkich możliwych rozwiązań a następnie sprawdzenie poprawności każdej z konfiguracji po kolei. Niestety metoda ta gwarantuje odnalezienie tylko bardzo prostych przypadków w rozsądnym czasie. W zaistniałych okolicznościach należało poszukać metody, która w podobny sposób dała by się zrównoleglić i przy okazji dawała możliwość obliczenia większego spektrum przypadków, w krótszym czasie. Wariacją na temat techniki Brute Force jest Backtracking. Działa w podobny sposób, z wyjątkiem:

• Poprawność wyprodukowanych wyników jest sprawdzana po każdym podjętym kroku.

• W przypadku braku wariantów postawienia następnego kroku, istnieje możliwość powrotu do poprzedniej decyzji i wybór innego wariantu.

W celu dodatkowego zoptymalizowania procesu wyszukiwania rozwiązania za pomocą Backtrakingu, można zastosować wcześniej przeszukiwanie wszerz (Breadth First Search). Algorytm ten w kontekście Sudoku pozwala na zminimalizowanie wariantów potencjalnego rozwiązania poprzez wyprodukowanie x alternatywnych diagramów, które predysponują do ostatecznego uzupełniena tablicy. Odnalezione przez BFS diagramy potencjalnego rozwiązania są przekazywane do metody realizującej Backtracking, gdzie nad

każdą tablicą pracuje osobny wątek. Wątki niezależnie od siebie wykonują pracę na różnych danych. Wątek, który jako pierwszy skończy pracę jest wybierany a obliczony przez niego wariant jest traktowany jako rozwiązanie. Pozostałe są odrzucane.

6 Implementacja

Szczegółowe etapy implementacji:

Na samym początku odbywa się sprawdzenie poprawności argumentów. Program oczekuje trzech argumentów wejściowych:

kernel.exe [liczba bloków] [liczba wątków w bloku] [ścieżka do pliku]

W następnym kroku pobierany jest plik wejściowego diagramu Sudoku. Struktura linii w pliku:

```
070065080
600030400
020040700
860002570
007406100
052300064
008020030
005080001
040710050
```

Linie powinny być przedstawiane w sposób ciągły, bez pomocy żadnych separatorów. Po każdej sekwencji dziewięciu znaków umieszczany jest znak nowej linii. Symbol 0 oznacza, lukę do wypełnienia.

Struktura diagramu jest reprezentowana w programie w postaci jednowymiarowej ciągłej tablicy, mogącej przechowywac początkowy diagram jak i produkty poszczególnych algorytmów. Aby uzyskać dostęp do konkretnej tablicy, należy dokonać przesunięcia: n * 81, dzie n to indeks konkretnej tablicy.

W następnej kolejności rejestrowane są zdarzenia CUDA a także alokowana pamięć dla wywołań kerneli.

Po przygotowaniu danych wejściowych i inicjalizacji wszystkich niezbędnych zasobów, sterowanie przejmuje metoda wykonująca algorytm BFS. Program zaczyna od pojedynczego wywołania kernela zawierającego implementację BFS. Do kernela przekazywane są dwie tablice. Źródłowa – do której wczytane została konfiguracja pierwszego diagramu oraz docelowa, do której zapisane zostaną produkty algorytmu. BFS skanuje diagram od początku, próbując odszukać następną lukę do uzupełnienia. Następnie w odnalezione miescje próbuje wstawić po kolei wszystkie możliwe wartości od 1 do 9 przy czym sprawdza czy bieżące wstawienie jest poprawne (zgodność kolumn, wierszy, podsekcji). Poprawne warianty pierwszego diagramu są zapisywane do tablicy docelowej. Jednocześnie zapamiętywana jest pozycja wszystkich pozostałych luk odnalezionego wariantu i ich liczba. Obie te informacje posłużą w późniejszych obliczeniach.

Po pierwszym wykonaniu kernela i uzyskaniu pierwszych alternatywnych diagramów, program wykonuje cykl osiemnastu iteracji poszukiwań dalszych wariantów. W tym celu tabela źródłowa i docelowa jest zamieniana referencjami – wyniki poprzedniego przeszukiwania stanowią teraz nowe dane wejściowe dla BFS. Operacje te są powtarzane aż do skończenia cyklu iteracji. Ich produktem jest graf potencjalnych rozwiązań. Warto zauważyć, że w algorytmie BFS zastosowany został mechanizm dywergencji. Pracuje co najwyżej

tyle wątków jednocześnie ile wariantów znajduje się w tablicy źródłowej. Oznacza to, że w przypadku pierwszego wywołania, pracował tylko jeden wątek.

Produkty poprzednich algorytmów są przekazywane do funkcji Backtrack. W przeciwieństwie do poprzedniego algorytmu, który wymagał cyklu iteracji, wywołanie tego kernela, od razu przydziela każdemu wątkowi osobny diagram do rozpatrzenia. W każdym wątku program korzysta z kolekcji zapamiętanych luk. Ustawia się na kolejnej i próbuje wstawić wartości od 1 do 9. Następuje sprawdzenie kolumn, wierszy i podsekcji. W przypadku gdy diagram jest określany jako poprawny, następuje przejście do kolejnej luki. W przypadku gdy, żadna z wartości od 1 do 9 nie zagwarantowała poprawności diagramu wskaźnik jest wycofywany do miejsca ostatniej uzupełnionej luki i następuje próba wpisania alternatywnej wartości. Wygrywa wątek, który jako pierwszy odszuka w całości poprawny diagram.

7 Wnioski

Algorytmy rozwiązywania Sudoku na GPU różnią się znacząco od klasycznych implementacji. Możliwość wykonania setek operacji jednocześnie oraz pewne ograniczenie technologii CUDA w stosowaniu rekursji determinują określoną manierę algorytmów. W zastosowanym podejściu ilość bloków/wątków dla optymalnego podziału zależy ściśle od rodzaju danych wejściowych. Najbardziej optymalne rozwiązanie zakłada powołanie co najwyżej takiej ilości wątków co liczba diagramów wyprodukowanych przez BFS. Przy większej liczbie wątków czas systematycznie się wydłuża z powodu kosztu ich powoływania. Zastosowane metody okazały się optymalne dla rozwiązań przypadków prostych, średniozaawansowanych i niektórch trudnych. Dla niektórych ektremalnie trudnych przypadków są niewystarczające.