Algorytm Smitha-Watermana - poszukiwanie optymalnych lokalnych dopasowań sekwencji

Michał Jaworek, Marcin Kaciuba
Politechnika Krakowska

12 stycznia 2015

Abstrakt

Algorytm dopasowania sekwencji polega na określaniu stopnia podobieństwa dwóch ciągów. Znajduje on swoje zastosowanie m.in. w bioinformatyce do poszukiwań dopasowań sekwencji nukleotydów i aminokwasów. Algorytm Smitha-Watermana rozwiązuje jeden rodzaj problemów tego typu - tzw. dopasowanie lokalne. W poniższym dokumencie przedstawiono opis algorytmu i jego zrównoleglenia z wykorzystaniem technologii CUDA.

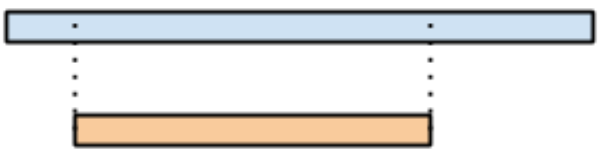
Przedostawienie problemu

Problem dopasowania sekwencji przyjmuje na wejściu dwa ciągi znaków. W ogólnym przypadku, ciągi te mogą składać się z liter dowolnego alfabetu. W przypadku zastosowań bioinformatyczych zazwyczaj ten alfabet jest relatywnie niewielki (np. czteroznakowy "TGAC").

Problem tej klasy można interpretować na dwa sposoby. Istnieją rozwiązania analizujące:

- dopasowanie globalne
- dopasowanie lokalne

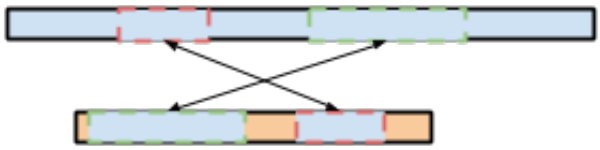
W przypadku dopasowania globalnego dwa ciągi porównywane są wzdłuż całej sekwencji. Takie rozwiązanie jest wykorzystywane przy analizie jednodomenowych białek. Algorytmem tego typu jest na przykład algorytm Needlemana-Wunscha. Schemat takiego dopasowania przedstawiono poniżej:



Rysunek 1: Schemat dopasowania globalnego

Lokalny typ dopasowania polega na rozszerzeniu możliwości algorytmów pierwszego typu o zdolność do zauważenia podobieństw w małych obaszarach. Dla przykładu pewne

sekwencje mogą być zamienione kolejnością. Ten typ rozwiązania znajduje zastosowanie w analizowaniu białek wielodomenowych. Poniżej przedstawiono schemat dopasowania tego typu.

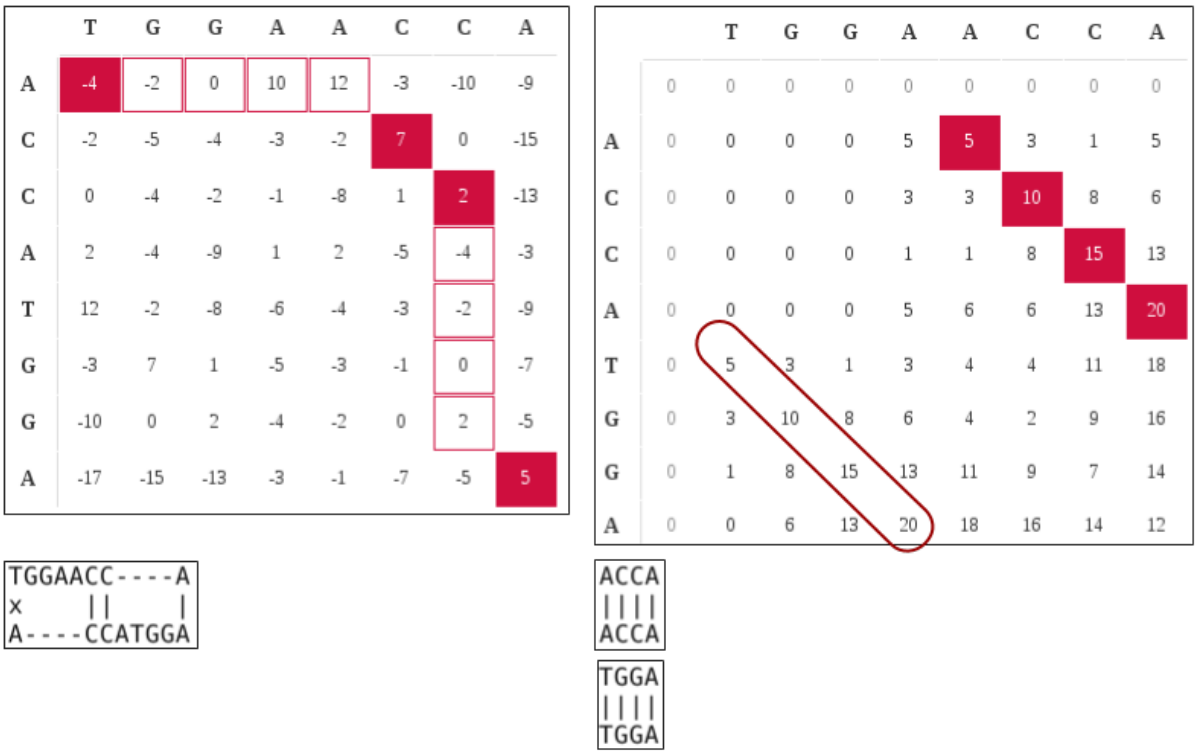


Rysunek 2: Schemat dopasowania lokalnego

Dla uwidocznienia różnicy w działaniu tych dwóch typów algorytmów posłużmy się przykładem następujących ciągów:

- TGGAACCA
- ACCATGGA

Powyższa sekwencja składa się z dwóch czteroliterowych sekwencji umieszczonych w różnej kolejności. Poniżej przedstawiono macierze podobieństawa uzyskane przez oba algorytmy wraz ze znalozionymi rozwiązaniami. Proces powstawania macierzy tego typu zostanie opisany w dalszej częście tego dokumentu.

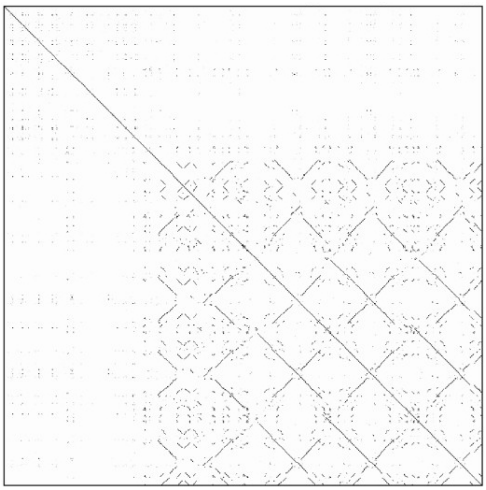


Rysunek 3: Przykład różnicy w działaniu dopasowania globalnego (po lewo) i dopasowania lokalnego (po prawo)

W przypadku globalnego dopasowania najlepszy uzyskany wynik jest jeden. Algorytm uzaje, że za najplepsze dopasowanie należy uznać następującą interpretację:

- Pierwszy znak został podmieniony
- Natępnie brakuje 4 znaków w drugim ciągu
- Kolejne dwa znaki pasują do siebie
- Natępnie brakuje 4 znaków w pierwszym ciągu
- Ostatnie znaki pasują do siebie

Jak widać, takie rozwiązanie nie jest w stanie wykryć istoty zadanego przykładu. Dla odmiany dopasowanie lokalne nie narzuca jednego najlepszego rozwiązania. Po zbudowaniu macierzy podobieństwa możemy zauważyć, że istnieją dwie ścieżki punktowane w ten sam sposób. Jedna z nich reprezentuje informacje o znalezieniu dopasowania podciągów ACCA, druga o znalezieniu dopasowania podciągów TGGA. W przypadku dużych ciągów wejściowych powyższe macierze reprezentuje się w odmienny sposób. Przyjmując pewną wartość progową można utworzyć wykres tego typu:



Rysunek 4: Przykład wizualnej reprezentacji macierzy podobieństwa dla algorytmu dopasowania lokalnego

Dzięki takiemu przedstawieniu wyników możliwe jest zwrócenie uwagi na fragmenty zawierające istotne podobieństwo.

Algorytm Smitha-Watermana w ujęciu sekwencyjnym

Algorym Smitha-Watermana należy do klasy algorytmów dynamicznych. Składa się z się z dwóch etapów:

- Tworzenie macierzy podobieństwa
- Otwarzanie optymalnej ścieżki (ang. backtracking)

W pierwszym etapie zostaje utworzona pusta macierz. Jej wiersze odpowiadają kolejnym znakom pierwszego ciągu, kolumny kolejnym znakom drugiego ciągu. Komórki znajdujące się na przecięcie opisują punktację określającą w jakim stopniu dopasowanie danych dwóch znaków jest poprawne.

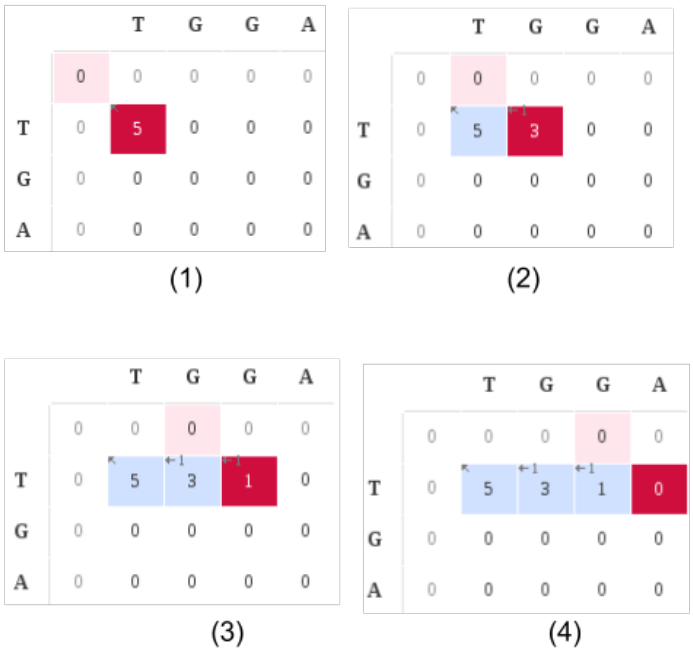
W celu wypełnienia powyższej macierzy należy zauważyć, że przy porównywaniu dwóch ciągów mogą mieć miejsce trzy sytuacje przedstawione na poniższym schemacie.

Rysunek 5: Możliwe sytacje w trakcie porównywania ciągów: (1 - dopasowanie, 2 - przerwa, 3 - zamiana)

Dwa ciągi są do siebie podobne gdy mamy więcej sytuacji typu 1 (dopasowanie) niż sytuacji typów 2 (przerwa), 3 (zamiana). W związku z tym spostrzeżeniem w trakcie wypełniania macierzy wartościami będziemy dodatnio punktować dopasowania, podczas gdy przerwy i zamiany będę punktowane ujemnie. Dokładne wartości punktacji nie są elementem specyfikacji algorytmu i są dobierane w zależności od rozpatrywanego problemu. Daje to możliwość porównywania ciągów w sposób traktujący przerwy mniej restrykcyjnie niż zamiany (lub odwrotnie).

W celu opisu kroków algorytmu posłużono się przykładem. Schemat przedstawiony na rysunku 6. prezentuje pierwsze kroki wykonywane w celu wypełnienia macierzy podobieństwa. Przedstawiono przypadek dla danych wejściowych: TGGA, TGA oraz dla parametrów:

- Punktacja za dopasowanie: 5
- Punktacja za zamianę: -3
- Punktacja za przerwę: -2



Rysunek 6: Pierwsze cztery kroki wykonania algorytmu Smitha-Watermana dla danych wejściowych: TGGA, TGA

Na początku macierz wypełniona jest zerami. Należy zwrócić uwagę na istnienie dodatkowego wiersza i kolumny (oznaczone na schematach szarą czcionką). Komórki rozpatrywane są wiersz po wierszu. Dla każdej komórki wykonywana jest sekwencja kroków mająca odpowiedzieć na następujące pytanie: "Przez którego z sąsiadów należy poprowadzić ścieżkę dopasowania tak, żeby w obecnej komórce osiągnąć najlepszy wyniki?". Przy odpowiedzi na to pytanie rozpatrywane są komórki:

- Na lewo - odpowiadająca wprowadzeniu przerwy w pierwszym ciągu
- Powyżej - odpowiadająca wprowadzeniu przerwy w drugim ciągu
- Na skos (powyżej i na lewo) - w zależności od przypadku odpowiadająca wprowadzeniu zamiany lub dopasowania.

Tą procedurę można przedstawić w pseudokodzie w następujący sposób.

```
int fromLeft = valueOfCellOnLeft - penaltyForGap;
int fromUp = valueOfCellOnUp - penaltyForGap;
int fromDiagonal;
if(letterInRow == letterInCol){
    fromDiagonal = valueOfCellInDiagonal + bonusOfMatch;
}else{
    fromDiagonal = valueOfCellInDiagonal + penaltyForReplacement;
}
int    valueOfThisCell = max(fromLeft, fromUp, fromDiagonal);
rememberDecision();
```

Prześledźmy kolejne kroki wykonania przykładu z rysunku 6. Warto przy tej okazji zwrócić uwagę na fakt, że w algorytmie Smitha-Watermana celowo unika się wprowadzania do macierzy ujemnych wartości. Ze względu na to w poniższym opisie zastosowano znak $\simeq$ wszędzie tam, gdzie zamiast wartości ujemnej podstawiane jest 0.

W kroku 1:

- Wybranie drogi z lewej strony dawałoby: 0 + (-2) ≃ 0
- Wybranie drogi z góry dawałoby : 0 + (-2) ≃ 0
- Ze względu na to, że litery w kolumnach (T) i rzędach (T) są takie same, przejście na skos dawałoby: 0 + 5 = 5
- Najbardziej opłacalnym ruchem jest przejście na skos, więc zapamiętujemy ten ruch i nadajemy komórce wartość 5

W kroku 2:

- Wybranie drogi z lewej strony dawałoby: 5 + (-2) = 3
- Wybranie drogi z góry dawałoby: 0 + (-2) ≃ 0
- Ze względu na to, że litery w kolumnach (G) i rzędach (T) nie są takie same, na skos dawałoby: 0 + (-3) ≃ 0
- Najbardziej opłacalnym ruchem jest przejście z lewej strony, więc zapamiętujemy ten ruch i nadajemy komórce wartość 3

W kroku 3:

- Wybranie drogi z lewej strony dawałoby: 3 + (-2) = 1
- Wybranie drogi z góry dawałoby : 0 + (-2) ≃ 0
- Ze względu na to, że litery w kolumnach (G) i rzędach (T) niesą takie same na skos dawałoby: 0 + (-3) ≃ 0
- Najbardziej opłacalnym ruchem jest przejście z lewej strony więc zapamiętujemy ten ruch i nadajemy komórce wartość 1

W kroku 4:

- Wybranie drogi z lewej strony dawałoby: 1 + (-2) ≃ 0
- Wybranie drogi z góry dawałoby: 0 + (-2) ≃ 0
- Ze względu na to, że litery w kolumnach (A) i rzędach (T) nie są takie same na skos dawałoby: 0 + (-3) ≃ 0
- W tym momencie wszystkie drogi dają taką samą wartośc, nie zapamiętujemy kierunku i nadajemy komórce wartość 0.

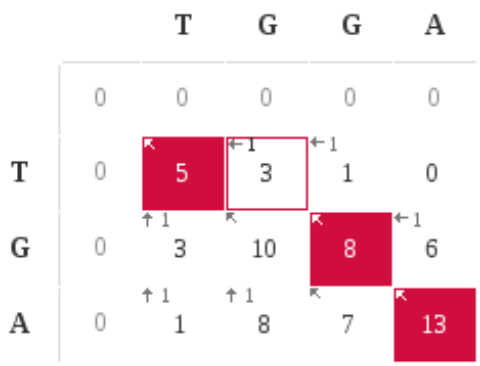
Po wykonaniu analogicznych kroków dla wszystkich komórek macierzy otrzymamy następujący stan:

		T	G	G	A
	0	0	0	0	0
T	0	5	3	1	0
G	0	3	10	8	6
A	0	1	8	7	13

Rysunek 7: Całkowicie wypełniona macierz dopasowania algorytmu Smitha-Watermana dla danych wejściowych: TGGA, TGA

W tym momencie macierz jest gotowa do wykonania drugiej fazy - backtrackingu. Polega ona na znalezieniu maksymalnej komórki i zapisaniu wszystkich kroków, które doprowadziły to ustalenie jej wartości. Poniżej przedstawiono macierz z naniesioną ścieżką metody backtrackingu.

Po wykonaniu analogicznych kroków dla wszystkich komórek macierzy otrzymamy stan przedstawiony na rysunku 8.



Rysunek 8: Backtracking dla algorytmu Smitha-Watermana dla danych wejściowych: TGGA, TGA

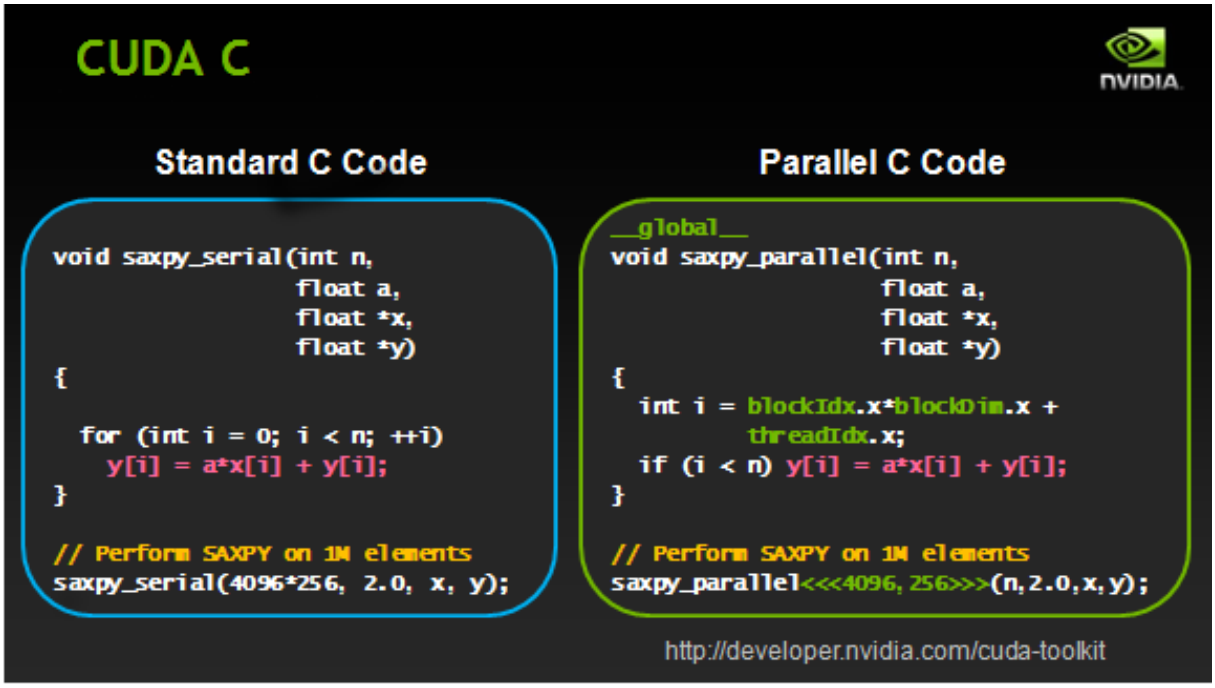
W efekcie uzyskane wyniki to: [↖, ←, ↖, , ↖] co należy odczytać jako: [dopasowanie, przerwa, dopasowanie, dopasowanie]. Odpowiada to dopasowaniu przedstawionym na rysunku 9.

Rysunek 9: Wynik działania algorytmu Smitha-Watermana dla danych wejściowych: TGGA, TGA

Zarys technologii CUDA

W ostatnich latach wzrost wydajności CPU był osiągany przez przez zwiększanie częstotliwości zegara, co jednak przestało być skuteczne. Zaczęto szukać rozwiązań polegających na zwiększaniu liczby rdzeni działających równolegle. W przypadku procesorów trend ten jest utrzymywany, jednak zauważono, że znaczące polepszenie wyników można także uzyskać stosując karty graficzne. Potokowy charakter obliczeń graficznych zaowocował powstaniem całej rodziny architektur wyspecjalizowanych w równoległym wykonywaniu tego typu zadań. W efekcie karty te zostały wyposażone w środowiska programistyczne dające możliwość wykonywania obliczeń o ogólnym charakterze. Jednym z takich rozwiązań jest CUDA (ang. Compute Unified Device Architecture) - standard opracowany przez firmę NVidia. Aktualnie karty graficzne posiadają setki rdzeni i bardzo szybką pamięć, co pozwala przy odpowiednio napisanym algorytmie na uzyskanie bardzo dużego przyspie-

szenia. Widać to na przykład w przypadku łamania haseł, dzięki obliczeniom na karcie graficznej czas metody bruteforce bardzo się skraca.



Rysunek 10: Porównanie progamu sekwencyjnego i równoległego napisanego w standardzie CUDA - przykład szkoleniowy z materiałów firmy NVidia

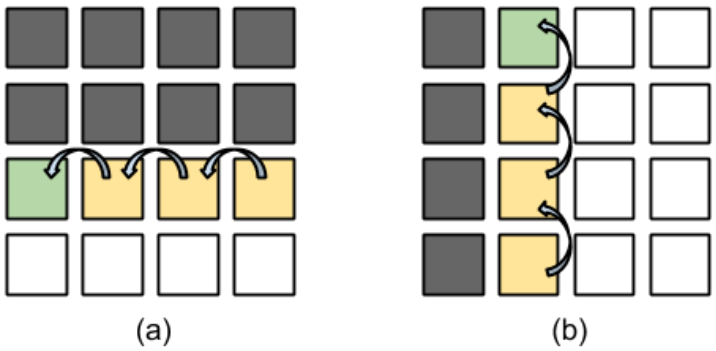
Przyspieszenie to nie jest tak łatwe do uzyskania, jak przykładowo w OpenMP, gdzie stosuje się dyrektywy. Aby uzyskać znaczne przyspiesznie na kartach graficznych należy odpowiednio przygotować algorytm. W przypadku uruchamiania go na GPU kod kernela (pojedynczej funkcji) wykonywany jest przez setki wątków. Jest to realizacja obliczeń typu SIMD z taksonomii Flyna.

Model zwrónoleglenia Algorytmu Smitha-Watermana

Zazwyczaj w algorytmach równoległych operujących na macierzach następuje podział obszarów w jeden z poniższych sposobów.

- Każdy wątek dostaję jedną kolumne lub jeden wiersz
- Każdy wątek dostaje jedną komórkę w kolumnie
- Każdy wątek dostaje jedną komórkę w wierszu

Żaden z powyższych modeli nie nadaje się do zrównoleglenia algorytmu Smitha-Watermana. Wynika to z zależności, które zostały symbolicznie przedstawione na rysunku 11.



Rysunek 11: Zależności występujące między komórkami w trakcie obliczeń algorytmu Smitha-Watermana

Na rysunkach 11, 12 przyjęto następujące oznaczenia:

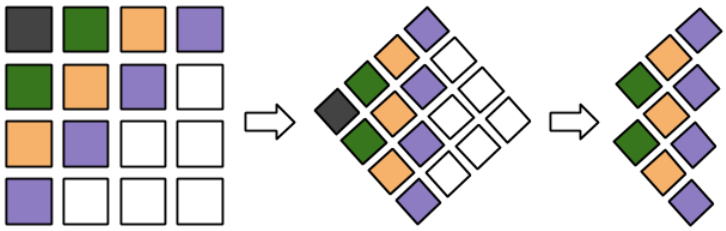
- komórki szare posiadają obliczoną wartość.
- komórki zielone mogą być obliczane bez zależności
- komórki pomarańczowe są zależne od wartości innych komórek
- komórki białe będę rozpatrywana w przyszłości

Ze względu na wspomniane zależności przy zrównoleglaniu algorytmu Smitha-Watermana stosuje się model polegający na podziale komórek należących do tych samych przekątnych. Schemat tego podziału przedstawia rysunek 12.

Rysunek 12: Prawidłowy model zrównoleglania algorytmu Smitha-Watermana

Jak widać taki podział umożliwia wielu wątkom wykonywanie operacji na swoich komórkach bez obawy o zależności.

Dodatkowo w celu zaoszczedzenia pamięci można posłużyć się kolejnym udoskonaleniem. Można zauważyć, że w celu obliczenia wartości komórek z n-tej przekątnej potrzebne są wartości z n-1 oraz n-2 przekątnej. Wcześniejsze komórki nie muszę być przechowywane. Rysunek 13. przedstawia w ten sposób oszczędności pamięci.



Rysunek 13: Metoda oszczędności pamięci przy wykonaniu algorytmu Smitha-Watermana

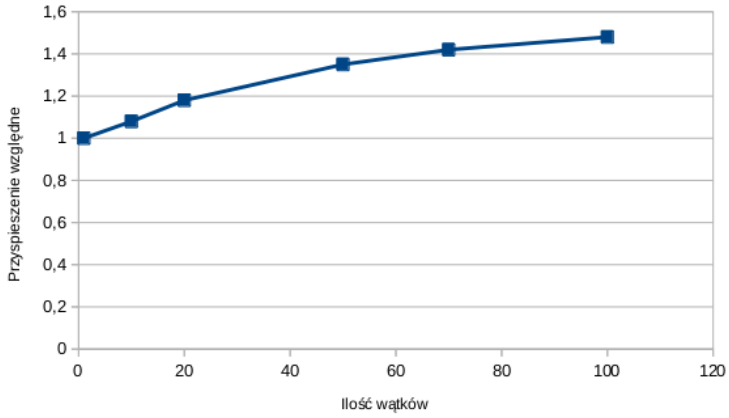
Obliczenia i wnioski

Dokonano prób implementacji powyżej opisanej metody. Wykorzystano sprzęt NVIDIA GeForce GT 430 z zestawem deweloperskim CUDA SDK w wersji 6.0.

Implementacja z wykorzystaniem technologii narzuca dodatkowe wymagania związane z

- prawidłową komunikacją między hostem a kartą graficzną
- prawidłowym rozmieszczeniem danych w stosownych obszarach pamięci

Realizacja zadania przebiegła pomyślnie tylko częściowo. Algorytm został przeniesiony na kod standardu CUDA i zwraca prawidłowe wyniki. Napotkano problemy z wydajnością. Algorytm przyspiesza w niewielkim stopniu. Poniżej przedstawiono wykres przspieszenia wyliczony ze wzoru $S_p = \frac{t_1}{t_{\|}}$



Rysunek 14: Przyspieszenie względne uzyskane przez program

Powyższe wyniki zostały uzyskane przez program uruchamiany dla danych wejściowych odpowiednio 13072 oraz 12960 znakowych. Prawdopodobnie kluczowym problemem przy zrównoleglaniu algorytmu jest liczne występowanie instrukcji warunkowych w trakcie obliczania punktacji komórek. Nie udało się znaleźć metody zastąpienia instrukcji if innymi instrukcjami nie zaburzającymi przyspieszenia.

Literatura

[1] Łukasz Ligowski,Witold Rudnicki *An efficien implementation of Smith-Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases.*

[2] E. Banachowicz *Bioinformatyka - wykład monograficzny*

[3] A. Skowron *http://opal.przyjaznycms.pl*