Wyższa Szkoła Technologii Informatycznych
w Katowicach

REFERAT PRACY DYPLOMOWEJ

Temat pracy: Implementacja i porównanie wydajności wybranych algorytmów grafowych w warunkach obliczeń równoległych

Autor pracy: Michał Podstawski

Promotor: Prof. WSTI, dr hab. Jarosław Śmieja

Kategorie: algorytmy grafowe

Słowa kluczowe: algorytmy grafowe, przetwarzanie równoległe, superkomputery

1. Cel i podstawowe założenia

Celem pracy była dostosowana dla potrzeb środowisk rozproszonych (przetwarzanie równoległe) implementacja algorytmów rozwiązujących cztery problemy grafowe – kolorowanie grafu (GC), poszukiwanie najkrótszej ścieżki z pojedynczego źródła (SSSP), znajdowanie minimalnego drzewa rozpinającego (MST) i liczenie trójkątów (TC). W ramach pracy poddano analizie istniejące rozwiązania teoretyczne, dokonując implementacji najbardziej obiecujących, a tam, gdzie było to konieczne, proponując również ich modyfikacje. W tych wypadkach, gdy żadne znane rozwiązanie nie przynosiło oczekiwanych wyników, przedstawiono nowe koncepcje.

2. Realizacja projektu

Wstępem do przygotowania implementacji było zapoznanie się z istniejącymi opracowaniami teoretycznymi dla wybranych algorytmów. Wzięto pod uwagę przede wszystkim rozwiązania powszechnie przyjęte za najlepiej rokujące – w tym kolorowanie grafu według pomysłu Erika Bomana oraz koncepcję Δ-stepping w wypadku poszukiwania najkrótszej ścieżki. Tam, gdzie brak było gotowych analiz teoretycznych dla algorytmów równoległych, wykorzystano istniejące algorytmy nierównoległe, próbując zmodyfikować je w taki sposób, by pracowały współbieżnie i dobrze skalowały się także dla potrzeb środowisk rozproszonych.

W trakcie pracy okazało się, że w rożnych algorytmach korzystna może okazać się zmiana kierunku aktualizowania wartości wierzchołka. Możliwe są kierunki: *pull* – kiedy procesowany wierzchołek aktualizuje swoją wartość, oraz *push* – kiedy aktualizuje wartości swoich sąsiadów. Żeby zbadać, dla których algorytmów takie zmiany kierunku mogą być korzystne, w każdym wypadku dokonano implementacji obydwu wariantów.

Wszystkich implementacji dokonano w języku C++ (w standardzie ISO/IEC 14882:2014), do obsługi procesowania współbieżnego używając biblioteki OpenMP w wersji 4.5. Podstawą systemu odwzorowywania grafów był *igraph* w wersji 0.7.1. Jako wsparcie wykorzystano bibliotekę Metis w wersji 5.1.0, a dodatkowo również Intel Threading Building Blocks 2017. Pomiary wykonano z użyciem frameworka LibSciBench.

W celu zebrania finalnych wyników użyto głównie grafów wybranych ze Stanford Large Network Dataset Collection oraz grafów R-MAT. Pełną informację o grafach przedstawiono w poniższej tabeli, gdzie *n* to ilość wierzchołków, a *m* ilość krawędzi:

TYP (SNAP)	**ID**	***n***	***m***
Social network (orc)	*orc*	3.07M	117M
Social network (pok)	*pok*	1.63M	30.6M
Ground-truth [4] community	*ljn*	3.99M	34.6M
Purchase network	am	262k	1.23M
Road network	*rca*	1.96M	2.76M
R-MAT	*rmat*	33M-268M	66M-4.28B

Jako środowisk uruchomieniowych użyto trzech maszyn o stosunkowo szerokim przekroju mocy obliczeniowych i architektur, co pozwala z bardzo wysokim prawdopodobieństwem oceniać stopień skalowalności algorytmów na maszynach HPC w ogóle; były to:

- *CSCS Piz Daint (Daint) - Cray XC30*: każda jednostka składowa posiada 8-rdzeniowy procesor Intel E5-2670 Sandy Bridge CPU z 32 GB DDR3-1600 RAM i NVIDIA Tesla K20X z 6 GB GDDR5 RAM; połączenia bazują na Cray's Aries i są implementacją topologii Dragonfly.

- *CSCS Piz Dora (Dora) - Cray XC40*: każda jednostka składowa posiada dwa 2-rdzeniowe procesory Intel Haswells E5-2690 i 64 GB DDR4 RAM; połączenia takie same jak w Daint.

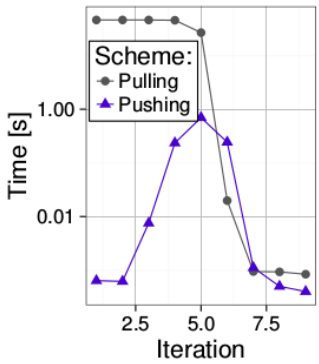
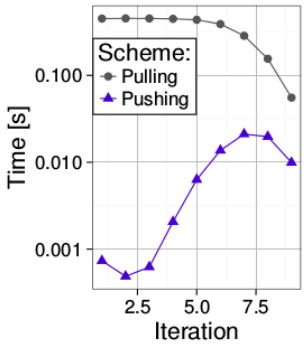
- *Trivium V70.05 (Trivium)*: posiada procesor Intel Core i7-4770, zawierający cztery 3.4 GHz Haswell z 2 wielowątkowymi rdzeniami, zapewniając sprzętowo 8 wątków; każdy rdzeń ma 32 KB cache L1 i 256 KB cache L2; procesor posiada 8 MB łączonego L3 cache i 8 GB RAM.

Lokalnie do programowania i testów używałem komputera z procesorem AMD A6-6310 (4 rdzenie, sprzętowo obsługa do czterech wątków; L1 cache 256 KB, L2 cache 2048KB) i 8 GB RAM.

3. Produkt końcowy

a) *SSSP – Single Source Shortest Path (najkrótsza ścieżka z pojedynczego źródła)*

Algorytm SSSP zaprogramowano zgodnie z teoretyczną koncepcją Δ-stepping. Δ-stepping jest algorytmem łączącym dobre strony dwóch najbardziej znanych synchronicznych rozwiązań tego problemu – algorytmu Djikstry i algorytmu Bellmana-Forda. Zgodnie z założeniami, dokonano implementacji w wariantach *push* i *pull*. Pomiary wskazały, że oba warianty dobrze sprawdzają się w środowisku rozproszonym, ale wyraźnie krótszy czas iteracji ma wersja *push*:

Analiza SSSP (XC30, *T* = *16*)	
a) *orc*	b) *am*
	

b) *GC – Graph Coloring (kolorowanie grafu)*

W wypadku kolorowania grafu dokonano trzech implementacji. Są do kolorowania *push* i *pull* zaprojektowane w oparciu o prace Erika Bomana, oraz dodatkowa implementacja, wykorzystująca podział całego grafu na mniejsze partycje – *buckety*.

Do implementacji kolorowania Bomana użyto synchronicznego kolorowania zachłannego (*greedy coloring*), które wykorzystano do równoległego pokolorowania części grafu, następnie ze sobą synchronizowanych. Wadą rozwiązania okazała się konieczność przechowywania w pamięci bardzo dużej struktury danych, jaką jest tablica oznaczająca kolory kolejnych wierzchołków.

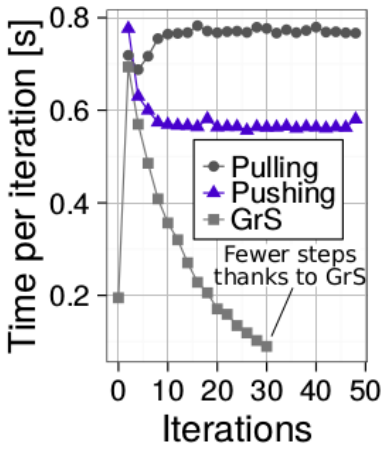
Z tego względu załączono trzecią implementację, która bazuje na analizie wierzchołków granicznych i przesuwaniu niepokolorowanych jeszcze wierzchołków do odpowiednich *bucketów*,

które oznaczają kolejny wolny kolor. Rozwiązanie to dało bardzo dobre rezultaty, a testy wskazują, że dla wszystkich zbadanych wypadków dostarczyło poprawnych wyników.

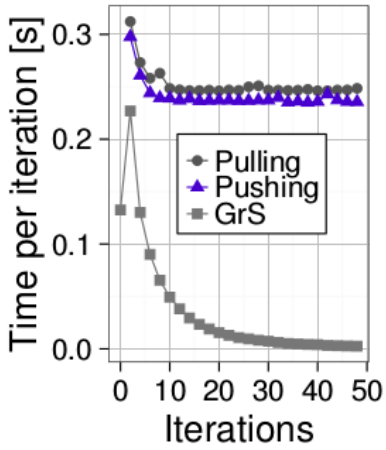
Pomiary długości iteracji dla zaimplementowanych algorytmów wyglądają następująco:

Analiza i porównanie algorytmów kolorowania grafu [GrS – wersja z *bucketami*] (Cray XC30, *T* = *16*)

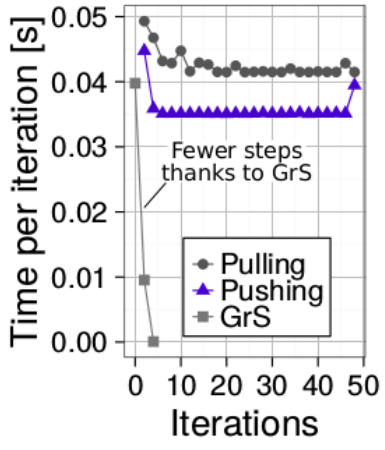
a) orc



b) ljn



c) rca



c) *TC – Triangle Counting (zliczanie trójkątów)*

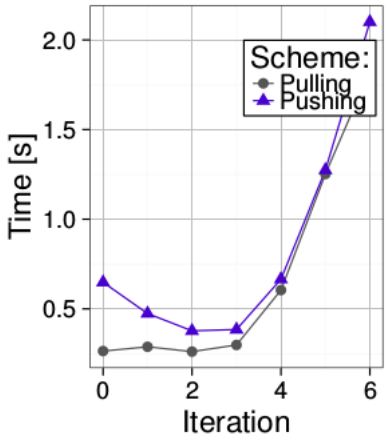
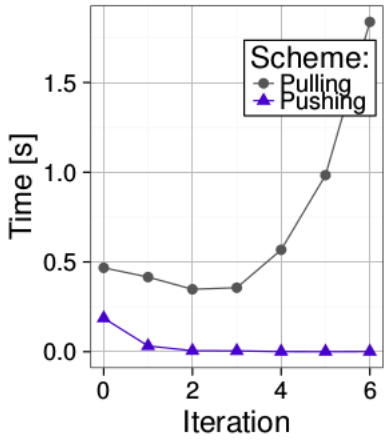
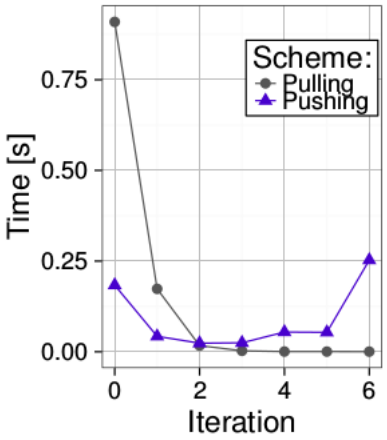
W tym algorytmie przyjęto, że interesuje nas ilość trójkątów, których częścią składową jest dany wierzchołek. Poszukiwania trójkątów dokonano w najbardziej klasyczny sposób – iterujemy po wszystkich wierzchołkach i ich sąsiadach, porównując pomiędzy nimi tablicę sąsiedztw (kluczowy moment algorytmu w kategoriach wydajności). Jeśli znajdziemy wspólnego sąsiada dla obu wierzchołków (i sąsiad ten jest różny niż oba badane wierzchołki), oznacza to, że wraz z nim stanowią one trójkąt. W tym wypadku korzystniejszy jest wariant *pull*, co można wywnioskować na podstawie osiągniętych czasów przebiegu algorytmu dla różnych grafów:

Wariant:	**TC – całkowity czas trwania w sekundach (Daint, XC30, *T* = *16*)**				
	orc	*pok*	*ljn*	*am*	*rca*
Push	11.78k	139.9	803.5	0.092	0.014
Pull	11.37k	135.3	769.9	0.083	0.014

d) *MST – Minimum Spanning Tree (Minimalne drzewo rozpinające)*

Do implementacji algorytmu szukającego minimalne drzewo rozpinające użyto klasycznej koncepcji Borůvki (Sollina). Okazuje się, że ten algorytm, przygotowany jako propozycja algorytmu synchronicznego, daje się stosować jako algorytm równoległy (wymagane są jedynie niewielkie zmiany). Kluczową trudnością jest znajdująca się w tym rozwiązaniu faza, w której zidentyfikowane wierzchołki (tzn. te, które już znalazły się w obrębie drzewa), łączone są w superwierzchołek, który od tego momentu stanie się bazą poszukiwań – w praktyce jest to najpoważniejsze wyzwanie, zarówno jeśli chodzi o projekt, jak i o wydajność algorytmu.
Poniżej przedstawiono czasy iteracji dla wariantu *push* i *pull*. W tym wypadku wariant *push* jest wyraźnie korzystniejszy (choć są pojedyncze iteracje, gdzie to *pull* wykonuje się szybciej):

Porównanie wydajności algorytmów MST w różnych fazach (XC40, graf *orc*, *T = 16*)

a) szukanie najlżejszej krawędzi	b) przygotowanie list do łączenia superwierzchołków	c) łączenie superwierzchołków
		

4. Informacje o możliwości wykorzystania / wykorzystaniu pracy

Wszystkie przedstawione implementacje charakteryzują się dobra wydajnością i skalowalnością, w sposób właściwy zachowując się w testowych środowiskach rozproszonych. Z tego względu mogą okazać się przydatne zarówno dla obecnych zastosowań praktycznych (takich jak szczegółowa analiza powiązań w sieciach społecznościowych, poszukiwania najkrótszej drogi), jak i jako wstęp do dalszych badań i analiz w tej dziedzinie.