Konrad Paziewski
306410

Maksymalna podtablica

PWIR 2014, zad 3

Opis rozwiązania:

Zaimplementowałem prosty sześcienny algorytm rozwiązujący problem maksymalnej podtablicy.

Każdej parze wierszy **(i, j)** (i <= j) w tablicy wejściowej zostanie przyporządkowany jeden proces. Proces ten ma za zadanie znaleźć maksymalną podtablicę tablicy wejściowej taką że jej pierwszym wierszem jest **i**, a ostatnim wierszem jest **j**. Wykorzystuje do tego algorytm Kadane'a.

Dokładniej:
Każdy proces przygotowuje tablicę wejściową niezależnie. Proces wczytuje tablicę. Następnie, jeżeli liczba wierszy w tablicy jest większa niż liczba kolumn tablica jest transponowana.
W tym momencie wiemy że liczba wierszy **M** tablicy jest mniejsza lub równa liczbie kolum **N**.
Załózmy że algorytm zostanie uruchomiony na **P** procesach.
Następnie, każdy proces generuje dwie pomocnicze struktury danych:

1. **rank_responsible[M][M]** – tablica taka, że **rank_responsible[i][j] = w** wtedy i tylko wtedy gdy za parę wierszy **(i,j)** odpowiada proces o randze **w**. Procesy nie komunikują się żeby ustalić **rank_responsible** – każdy proces wylicza je niezależnie w oparciu o rozmiar tablicy i liczbę dostępnych procesów. Obliczenie jest deterministyczne więc każdy proces wygeneruje identyczne **rank_responsible**. Każdy proces dostanie podobną ilość par do rozpatrzenia.
2. **partial_columns[M][N]** – **partial_columns[i][c]** to suma wszystkich elementów z tablicy wejściowej w **i**-tym (pod względem długości) prefiksie kolumny **c.** Zauważmy że problem znalezienia maksymalnej podtablicy dla wiersza początkowego **i** oraz końcowego **j** sprowadza się do uruchomienia algorytmu Kadane'a na tablicy **partial_columns[j] – partial_columns[i – 1].**

Proces szuka w **rank_responsible** par wierszy za które jest odpowiedzialny i uruchamia na nich algorytm Kadane'a. Wybiera maksimum ze wszystkich wartości które rozpatrzył i wysyła to maksimum do procesu o randze **0**. Proces o randze **0** odbiera wyniki częściowe i wybiera z nich maksimum (liniowo). Zatem wysłanych/odebranych zostanie dokładnie **P – 1** komunikatów.

Analiza:
Rozpatrzmy czas działania procesu **0**. Początkowy preprocessing jest kwadratowy. Policzenie **rank_responsible** w mojej implementacji zajmuje **O(M^2)**, **partial_columns** natomiast **O(MN)**. Sam algorytm wymaga **O(M^2)** operacji na przejście po **rank_responsible** i **O(NM^2 / P)** operacji na algorytm Kadane'a na wszystkich **O(M^2/P)** parach wierszy za które odpowiedzialny jest proces. Ostatecznie proces **0** zbiera wyniki w czasie **O(P).** Ostecznie zatem czas działania procesu 0 dany jest wzorem **O(M^2 + MN + NM^2/P + P) = O(MN + NM^2/P + P)**, bo **M <= N**. Czasy działania pozostałych procesów nie zawierają ostatniego składnika **P**. Można zauważyć zatem, że algorytm działa lepiej dla małych wartości **P**. Wtedy czas działania można oszacować jako **O(NM^2/P)**. Jeżeli jednak liczba procesów będzie rosła to czas preprocessingu – który nie zależy od liczby procesów – może stać się dominującą częścią algorytmu. Dodatkowo wraz ze wzrostem liczby procesów będzie rosnąć narzut na proces **0** związany z wyborem maksimum. Ten problem można by częściowo zneutralizować wykorzystując algorytm rozproszonego wyboru maksimum. Wtedy składnik **P** można

by zastąpić składnikiem **log P**. W przypadku tak niewielkiej liczby procesów i stosunkowo dużej ilości liczby wierszy na jakich aglorytm miał działać problemy te nie miały praktycznego znaczenia na wydajność.

Implementacja:

W msp-par.c można odkomentować definicję makra **DEBUG** – dzięki temu program zacznie wypisywać większa ilość informacji – między innymi czasy działania poszczególnych fragmentów algorytmu. Funkcjonalność ta nie jest dobrze przetestowana ale nadal pomocna.
Wywołanie **MPI_Barrier** tuż po wygenerowaniu tabeli wejściowej ma na celu zsynchronizowanie procesów przed rozpoczęciem algorytmu. Nie ma to znaczenia dla poprawności algorytmu, zapewnia jednak że w momencie gdy zaczniemy liczyć czas wszystkie procesy będą już miały tablicę wejściową. Do mierzenia czasu użyłem funkcji **gettimeofday**.

Testy poprawnościowe:

Poprawność algorytmu testowałem na własnej instalacji MPI na jednej maszynie z czterordzeniowym procesorem i7. Skrypt testowy **test.hs** wykonuje 100 losowych niewielkich testów zarówno na mojej implementacji **mpi-par.exe** jak i na dostarczonej z treścią zadania implementacji **mpi-seq-naive.exe**. Następnie porównuje wyniki i raportuje ewentualne błędy. Jako że skrypt porównuje wyjścia programów, mogłaby nastąpić sytuacja gdy poinformowałby o błędzie, chociaż bład by nie nastąpił – kiedy istnieje więcej niż jedna maksymalna podtablica. Na studentsie może brakować bibliotek niezbędnych do uruchomienia tego skryptu (*cabal install regex-compat*).

Testy wydajnościowe:

Testy wydajnościowe przeprowadzilem (zgodnie z treścią zadania) na klastrze **notos** w kilku konfiguracjach. Testy zostały przeprowadzone na macierzach o rozmiarze 4000 x 4000. Czasy podane w sekundach. Podane wyniki są oparte o czasy działania procesu 0. Wyniki z dokładnością do trzech miejsc po przecinku:

Proc	Nodes	Max	Min	Avg	Comm
1	1	692.200	692.200	692.200	0.000
2	1	346.773	346.760	346.769	0.457
2	2	346.757	346.757	346.757	0.457
4	1	174.076	174.075	174.075	0.317
4	2	174.056	174.043	174.051	0.303
4	4	174.041	174.041	174.041	0.316
8	2	87.576	87.540	87.552	0.237
8	4	87.508	87.507	87.508	0.223
8	8	87.508	87.508	87.508	0.237
16	4	44.228	44.226	44.227	0.150
16	8	44.196	44.186	44.193	0.146
16	16	44.197	44.196	44.196	0.149
32	8	22.535	22.535	22.535	0.084
32	16	22.508	22.505	22.506	0.082
64	16	11.685	11.681	11.683	0.046

- Proc – liczba wszystich procesów
- Nodes – liczba maszyn
- Max – maksymalny czas z trzech uruchomień danej konfiguracji
- Min – minimalny czas z trzech uruchomień danej konfiguracji
- Avg – średnia z czasów trzech uruchomień danej konfiguracji
- Comm – średni czas jaki proces **0** spędzał w funkcji **gather_max** – odpowiada to komunikacji/synchronizacji z innymi procesami

Zauważmy że widać pewną korelację między wartościami w kolumnach Comm i Proc – czas komunikacji zależy od liczby procesów ale nie w sposób rosnący. To nie jest zgodne z moimi oczekiwaniami.

Proc	Nodes	Speedup	Effect	Seq speedup	Seq effect
1	1	1.000	1.000	4.026	4.026
2	1	1.996	0.998	8.037	4.018
2	2	1.996	0.998	8.037	4.018
4	1	3.976	0.994	16.010	4.002
4	2	3.977	0.994	16.012	4.003
4	4	3.977	0.994	16.013	4.003
8	2	7.906	0.988	31.831	3.979
8	4	7.910	0.989	31.847	3.981
8	8	7.910	0.989	31.847	3.981
16	4	15.651	0.978	63.013	3.938
16	8	15.663	0.979	63.062	3.941
16	16	15.662	0.979	63.057	3.941
32	8	30.716	0.960	123.667	3.865
32	16	30.756	0.961	123.828	3.870
64	16	59.246	0.926	238.531	3.727

- Proc, Nodes – bez zmian
- Speedup – speedup względem mojego rozwiązania w konfiguracji jednoprocesowej
- Effect – efektywność względem mojego rozwiązania w konfiguracji jednoprocesowej
- Seq speedup, Seq effect – analogicznie jak Speedup i Effect ale względem średniej wyników sekwencyjnych załączonych do treści zadania

Wraz ze wzrostem liczby procesów efektywność spada – smutne aczkolwiek zgodnie z oczekiwaniami. Stały (względem liczby procesów) czas preprocessingu tabeli zaczyna mieć coraz większe znaczenie dla czasu działania całego rozwiązania.