Звіт з проекту Ad Fontes
з курсу
Алгоритми та структури даних

Алгоритм:
“Bubble Scheduling: A Quasi Dynamic Algorithm for Static Allocation of Tasks to Parallel Architectures ”

Підготували:
Єлісєєв Юрій
Міліщук Роман

У даній роботі ми маємо на меті порівняти алгоритм BSA(Bubble scheduling algorithm)
(http://ranger.uta.edu/~iahmad/conf-papers/%5BC26%5D%20Bubble%20scheduling%20A%20quasi%20dynamic%20algorithm%20for%20static%20allocation%20of%20tasks%20to%20parallel%20architectures%20.pdf)з двома іншими алгоритмами, завданням яких є також розподілення завдань із залежностями

Ми використали heft алгоритм (https://github.com/samanjate/heft), який також бере DAG (Directed acyclic graph) для представлення залежності між завданнями. Час роботи завдання execution time (подібно до computation cost у Bubble scheduling), а час передавання даних communication time (communication cost у Bubble scheduling). Даний алгоритм спочатку вираховує середній час роботи завдань, а також середній час передавання даних. Потім, на основі цих даних визначається рейтинг кожного завдання, сортується у порядку спадання. Тобто, можемо бачити, що це жадібний алгоритм, а тому не завжди даватиме оптимальну відповідь.

Також, для того, щоб порівняти, наскільки великий приріст дає багатоядерна система, використаємо визначення всього на одне ядро. Для цього використаємо CPN-First Ordering, який Critical Path Method, тобто визначає шлях, який має найбільшу важливість, а потім сортує вершини у порядку, що спочатку потрібно обробляти саме їх, потім вершини, які ведуть у Criticial Path, а в кінці всі решта.

Для порівняння ми використали дані, які були спочатку випадково згенеровані бібліотекою daggen (https://github.com/frs69wq/daggen), проте вона є досить гнучкою і дозволяє задати багато параметрів, чим ми й скористалися, а саме:

- n для вибору кількості вершин
- density для вибору розрідженості графа
- ccr (communication-to-computation ratio)

Ми поділили дані на такі класи:

- Повні - density=1.0
- Розріджений - density=0.3
- Дуже розріджений - density=0.01

Далі для кожного з цих даних ми брали різний ccr(0.1, 1.0, 10.0)

Так, як HEFT здатний розподілити на процесори лише, коли всі процесори з'єднанні, то будемо порівнювати лише на таких топологіях.

хз

Спочатку поглянемо, на скільки відсотків дає приріст BSA, відносно одного процесора при кількості завдань рівної 500

Density-ccr Кількість процесорів	1.0 0.01	1.0 1.0	1.0 10.0	0.3 0.01	0.3 1.0	0.3 10.0	0.01 0.01	0.01 1.0	0.01 10.0
2	71.46	58.95	57.96	68.42	79.42	69.48	64.57	76.66	83.6
4	232.	227.4	19.53	265.1	228.4	134.7	290.3	264.0	-91.68

	6								
8	517.1	428.3	390.8	588.3	479.5	350.6	620.4	350.7	450.6

Далі порівняємо його при таких самих параметрах з HEFT

Density-ccr Кількість процесорів	1.0 0.01	1.0 1.0	1.0 10.0	0.3 0.01	0.3 1.0	0.3 10.0	0.01 0.01	0.01 1.0	0.01 10.0
2	-69.76	-71.72	-74.54	-69.05	-67.78	-71.07	-74.15	-68.96	-67.55
4	-49.62	-45.12	-80.86	-47.39	-60.95	-47.0	-38.59	-98.56	-46.06
8	-98.8	-99.67	-97.32	-99.94	-95.15	-99.74	-99.95	-98.8	-99.91

Перевага HEFT в тому що йому не потрібна інформація про маршрутизацію, в той час коли BSA займається локальною оптимізацією. HEFT має лише відсортувати масив і розподілити його між процесорами. Через це, якщо використовувати архітектури з меншою кількістю процесорів, то алгоритм не буде витрачати багато часу на відбір і розподілення робіт між процесорами. Також, чим менше ccr, тим краще алгоритм краще працюватиме, адже тоді час на переміщення менш затратний і алгоритм може краще розподілити завдання. Також, можемо бачити, що алгоритм краще працює, коли граф більш розріджений, адже тоді менше залежностей і можна розподілити на на більшу кількість процесорів.

Наша реалізація BSA - https://github.com/Momka45/Bubble-Scheduling