Отчет по практическому заданию по курсу
«Параллельная обработка больших графов»

Параллельная реализация алгоритма дельта-степпинг решения задачи SSSP в ориентированном графе с весами

Выполнила: Домрачева Дарья, 523 группа
Вариант реализации: библиотека charm4py

Москва, 2021

Вычислительная система

Запуски производились на суперкомпьютере “Ломоносов 2”.

Основные технические характеристики суперкомпьютера "Ломоносов-2"	
Пиковая производительность	5,505 Пфлопс
Производительность на тесте Linpack	2,478 Пфлопс
Число вычислительных узлов	1 722
Основной тип процессора	Intel Haswell-EP E5-2697v3, 2.6 GHz, 14 cores
Тип ускорителя	NVidia Tesla K40M
Общее число ядер	64 384
Оперативная память на узел	64 GB
Основная сеть	Infiniband FDR
Сеть I/O	Infiniband FDR
Управляющая сеть	Gigabit Ethernet
Операционная система	CentOS 7

Выбор в пользу “Ломоносов 2” был сделан из-за проблем с установкой необходимых для программы библиотек на других доступных суперкомпьютеров. Стоит отметить, что и выбранная вычислительная система на данный момент не может поддерживать все возможные конфигурации charm4py из-за невозможности установить некоторые библиотеки (библиотеки должны быть установлены администраторами суперкомпьютера для их корректной работы).

Описание решения

Перед запуском программы производилась генерация графов с помощью предоставленного набора программ на языках C/C++ (в частности, gen_RMAT.cpp).

Подготовительный этап работы программы — создание массива объектов Chare (процессы), которые и будут обрабатывать входной граф. Граф хранится распределенно, принадлежность вершины процессу определяется следующим образом:

```
self.lg_size = 0
while 1 << self.lg_size != n_procs:
      self.lg_size += 1
      self.owner_const = (1 << self.lg_size) - 1
vertex_owner = vertex_global & self.owner_const
```

Процессы хранят информацию о локальных вершинах и ребрах (с весами) между ними, а также о ребрах, ведущих от локальных вершин к вершинам соседних процессов (CSR формат). В целях дополнительной оптимизации для хранения данных о графк использовались массивы numpy.

Перед началом работы основного алгоритма SSSP вычисляется значение delta, которое будет использовано процессами.

Перед началом пересылок все процессы делают основной шаг алгоритма и независимо друг от друга обновляют расстояния обработанных вершин.

Библиотека charm4py поддерживает вызовы reduce и allreduce. По своей сути они ближе к функциональности MapReduce, чем к reduce-операциям MPI, что позволяет предобрабатывать данные при вызове операций, а затем применять к ним необходимую функцию (базовые функции, предоставляемые библиотекой, включают min, max, gather и некоторые бинарные операции).

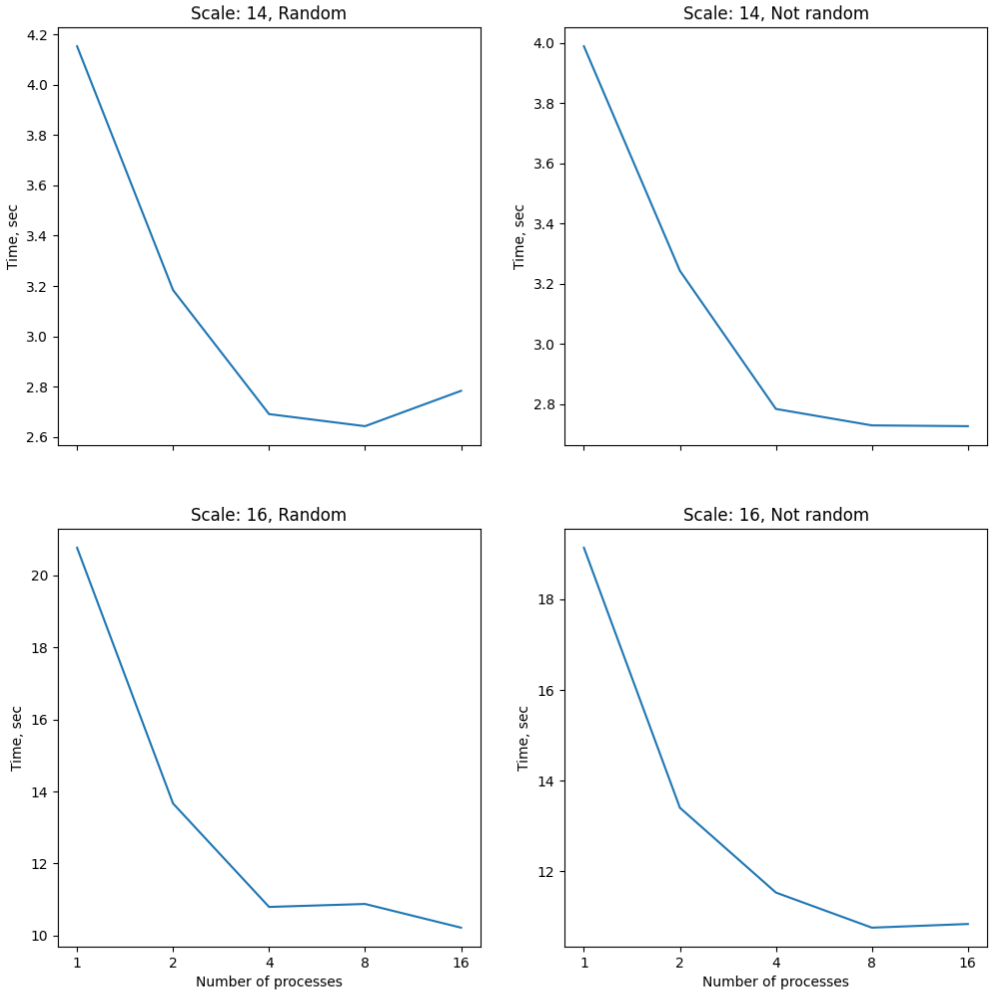
reduce-операции использовались при выборе индекса бакета для следующей итерации, а также для финального сбора минимальных дистанций со всех процессов после обработки всех бакетов.

Для синхронизации работы процессов (т.е. отслеживания момента окончания работы алгоритма) используется механизм Futures, которые могут возвращать какое-то значение только по окончании работы всех созданных процессов. Функции allreduce, использованные в реализации, как раз используют данный механизм.

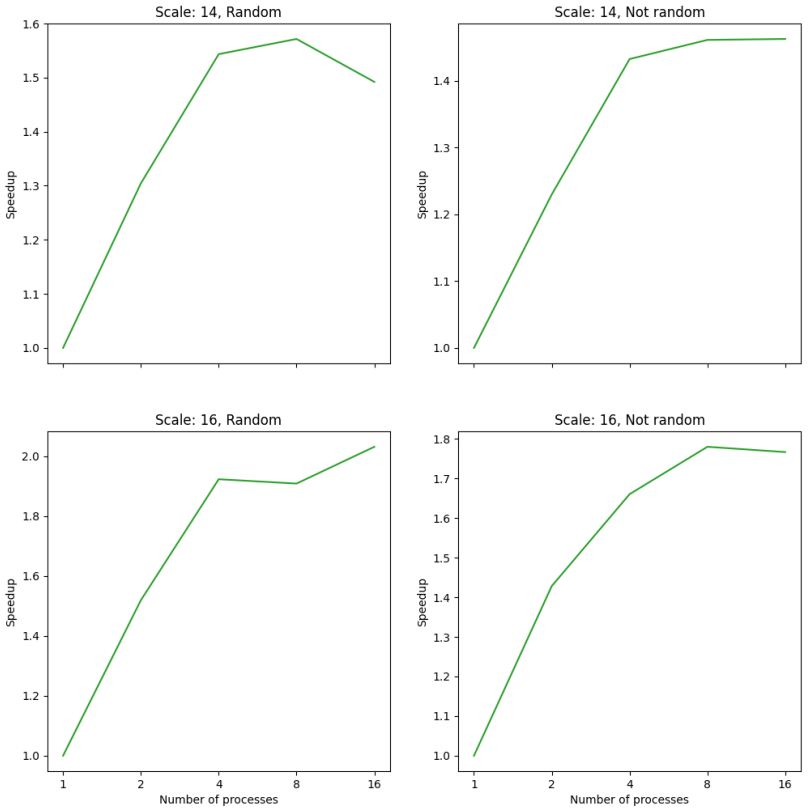
Для передачи данных об изменениях дистанций вершин, не являющихся локальными для некоторого процесса, был использован механизм отправки сообщений через каналы. Перед началом работы алгоритма каждый процесс определял своих “соседей” (те процессы, имеющие вершины, инцидентные некоторым локальным вершинам текущего процесса) и создавал набор каналов для общения с этими процессами. На каждой итерации процесс накапливал данные об изменениях дистанций, которые необходимо передать соседям, и после выполнения обработки бакета передавал одно сообщение со всеми данными об изменениях.

Результаты

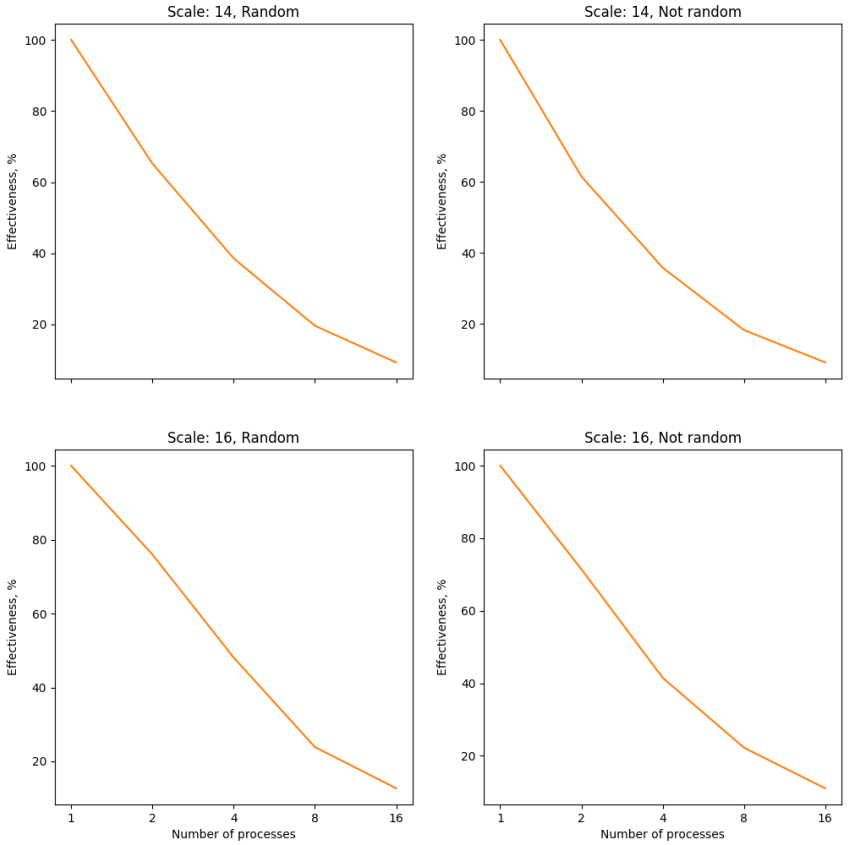
На графиках ниже представлены усредненные результаты времени работы программы, запущенной на 1, 2, 4, 8 и 16 процессах. Размеры графов: 2^{14} и 2^{16} вершин, средняя степень вершин — 40 и 50 соответственно. Тестировались два типа графов (с рандомизацией и без).



Полученное ускорение программной реализации представлено на графиках ниже.



Полученная эффективность (в процентах) программной реализации представлена ниже.



Вывод

На рассмотренных графах (имеются в виду графы, для которых представлены графики в разделе “Результаты”) была получена масштабируемость, однако эффективность распараллеливания снижается по мере увеличения числа процессов. Для графов размерности 2^{14} в какой-то момент время работы программы перестает значительно уменьшаться (примерно по достижению числа процессов равного 8), но в случае графов размерности 2^{16} этот эффект менее выражен, из чего можно преположить, что по мере увеличения размерности графа разработанная реализация будет показывать достаточно неплохую эффективность распараллеливания.
Стоит отметить, что заметное ускорение может быть получено уже на графах размерности 2^{13}.

Приложение

Проверка корректности представлена в виде лог-файла log.txt.