Параллельные вычисления

Параллельная реализация операций с сеточными данными на неструктурированной смешанной сетке

Лазарев Владимир Александрович

528 группа

17.10.2020

Описание задачи и программной реализации

Краткое описание задания

Целью задания является построения портера разреженной матрицы на основе представления неструктурированной сетки, построение СЛАУ на основе полученного портрета и решения СЛАУ.

Краткое описание программной реализации

В ходе разработки программы были реализованы следующие методы:

- *void createNodesOfGraph* – метод генерации портрета матрицы. Присутствует распараллеливание. Аргументами метода являются:
  - *int Nx* – ширина сетки;
  - *int Ny* – высота сетки;
  - *int k1* – количество несоединенных вершин графа;
  - *int k2* – количество соединенных вершин графа;
  - *map<int, vector<int>> resultGraph* – представление графа в виде справочника. Для каждого элемента по его индексу можно получить список его соседей;
  - *vector<int> IA, JA* – портрет матрицы;
  - *int sizeOfResultGraph* – количество вершин.
- *string vectorToString* – метод выводит на печать в консоль структуру типа vector<int>;
- *vector<string> createAdjacencyList* – метод выводит в консоль список смежности для каждой вершины;
- *void printResult* – печать результатов полученного портрета;
- *void makeSLAE* – метод построения СЛАУ на основе ранее созданного портера матрицы. Выполняется расчет ненулевых коэффициентов и вектора правой части. Присутствует распараллеливание. Аргументы:
  - *vector<int> IA, vector<int> JA* – портрет матрицы;
  - *vector<double> A* – ненулевые коэффициенты матрицы;
  - *vector<double> b* – вектор правой части;
  - *int countOfNodes* – количество вершин графа.
- *void printSLAE* – печать ненулевых коэффициентов каждой строки и значения вектора правой части;

- *double scalar* – принимает на вход две структуры типа vector<double> и рассчитывает их скалярное произведение (скалярное произведение векторов). Присутствует распараллеливание. Аргументы:
    - *vector<double> x1, x2* – вектора для перемножения;
    - *double allTime* – общее время, потраченное на текущий метод;
    - *double countOfCalls* – количество раз, сколько вызывался текущий метод.
- *double normalizeVector* – принимает на вход один вектор и возвращает его норму. Присутствует распараллеливание.
- *vector<double> spMV* – реализация матрично-векторного произведения. Присутствует распараллеливание. Список аргументов:
    - *int countOfNodes* – количество вершин графа;
    - *vector<int> IA, JA* – портрет матрицы;
    - *vector<double> A* – ненулевые коэффициенты матрицы;
    - *vector<double> x* – вектор, на который умножать;
    - *double allTime* – общее время, потраченное на текущий метод;
    - *double countOfCalls* – количество раз, сколько вызывался текущий метод.
- *vector<double> linearCombination* – реализация линейной комбинации двух векторов. Присутствует распараллеливание. Содержит следующие аргументы:
    - *vector<double> x1, x2* – вектора для кобинации;
    - *double a1, a2* – коэффициенты линейной комбинации;
    - *double allTime* – общее время, потраченное на текущий метод;
    - *double countOfCalls* – количество раз, сколько вызывался текущий метод.
- *void createMatrixFromA* – создание матрицы *M*, состоящей только из элементов главной диагонали матрицы *A*. Присутствует распараллеливание.
- *void reverseMatrix* – возведение матрицы в степень (-1). Получение обратной матрицы иными словами.
- *void solveSLAE* – реализация алгоритма решения СЛАУ итерационным методом с использованием метода сопряженных градиентов с предобуславливателем Якоби. Имеет следующие агрументы:
    - *vector<int> IA, JA* – портрет матрицы;
    - *vector<double> A* – ненулевые коэффициенты матрицы;
    - *vector<double> b* – вектор правой части;
    - *int countOfNodes* – количество вершин графа;
    - *double tol* – требуемая точность решения;

    - *int n* – количество итераций до получения решения;
    - *double res* – L2 норма невязки;
    - *vector<double> xRes* – вектор *x*, при котором удалось получить решение необходимой точности.
- *void printSolveVector* – печать вектора-решения СЛАУ *x*;
- *void doTask* – метод, выполняющий этап за этапом. Предполагалось, что можно вызвать несколько таких друг за другом с разным количеством потоков для наглядного представления преимуществ распараллеливания;
- *int main* – входная точка программы с инициализацией всех необходимых переменных.

Параметры запуска программы

Программа запускается со следующими параметрами:

1. *Nx* – ширина сетки. Целое число.
2. *Ny* – высота сетки. Целое число.
3. *k1* – количество несоединенных вершин графа. Целое число.
4. *k2* – количество соединенных вершин графа. Целое число.
5. *T* – количество нитей (поток), которые участвуют в распараллеленных участках программы. Целое число.
6. *tol* – необходимая точность решение СЛАУ. Число с плавающей точкой.
7. *isPrint* – необходимость печати детальной информации (таблица смежности, портрет матрицы, вектор решения СЛАУ). 0 – печать отключена, 1 – печать включена.

Опробованные методы оптимизации

В ходе выполнения практической работы возникали трудности с потреблением памяти и времени работы. Возникали проблемы, т. к. изначально вершина графа являлась объектов, хранящей вектор своих соседей (также объекты). В итоге программа отказывалась работать на размерах сетки более 15*15. Было принято решение реорганизовать вектор соседей вершины. Вместо объекта стал хранить всего лишь его индекс. Это позволило запускать программу на большем размере сетки. Заметив такой прирост производительности, убрал совсем объекты из программы, заменив на *map<int, vector<int>>*. Это также улучшило потребление памяти и скорость работы приложения. Детальнее рефакторинг кода можете наблюдать в репозитории

https://github.com/vlazarew/Parallel_Lab_1_Graph. При дальнейшем тестировании программы проведена оптимизация переменных, многие большие вектора были упразднены, чтоб во время выполнения в памяти хранились только необходимые данные без устаревших и утративших актуальность. Это позволило существенно уменьшить требуемую память для успешной работы приложения.

Исследование производительности

Характеристики вычислительной системы

ЭВМ 1 (ПК):

- Процессор – Intel Core i7-7700K;
- Количество ядер – 4;
- Базовая тактовая частота – 4,2 GHz;
- Максимальная тактовая частота – 4,5 GHz;
- GFLOPS – 249,6;
- Кеш-память – 8 MB;
- Частота системной шины – 8 GT/S;
- Количество общей оперативной памяти – 16 GB;
- BW – 2666 * 1* 8 = 21 Гб/с.

ЭВМ 2 (Ноутбук):

- Процессор – Intel Core i5-8520U;
- Количество ядер – 4;
- Базовая тактовая частота – 1,6 GHz;
- Максимальная тактовая частота – 3,4 GHz;
- GFLOPS – 163,2;
- Кеш-память – 6 MB;
- Частота системной шины – 4 GT/S;
- Количество общей оперативной памяти – 4 GB;
- BW – 1866 * 1* 8 = 15 Гб/с.

Описание параметров компиляции:

Используется компилятор *g++*, проект собирается под стандарт *C++ 11* с флагами *-O3, -fopenmp* и *-g.*

AI для линейной комбинации

```
for (int i = 0; i < vectorSize; i++)
        {
            result.at(i) = x1.at(i) * a1 + x2.at(i) * a2;
        }
```

Число флопов – 3N. 2 операции умножения + 1 операция сложения.

N даблов на чтение вектора *x1*. N даблов на чтение *x2*. 2 дабла на чтение *a1* и *a2*. N даблов на запись в вектор *result*. 1 дабл = 8 байт. Итого: 8*(3N + 2).

Получаем AI для линейной комбинации: $\frac{3N}{8*(3N+2)}$.

AI для скалярного произведения

```
for (int i = 0; i < vectorSize; i++)
        {
            result += x1.at(i) * x2.at(i);
        }
```

Число флопов – 2N -1. Операция умножения + сложение.

N даблов на чтение вектора *x1*. N даблов на чтение *x2*. 1 дабл = 8 байт. Итого: 8*2N.

Получаем AI для скалярного произведения: $\frac{2N-1}{8*2N}$.

AI для матрично-векторного произведения

```
for (int i = 0; i < countOfNodes; i++)
        {
            for (int k = IA.at(i); k < IA.at(i + 1); k++)
            {
                result.at(i) += A.at(k) * x.at(JA.at(k));
            }
        }
```

Число флопов – 2N.

N даблов на чтение вектора *A*. N даблов на чтение *x*. N даблов на чтение *IA*. N даблов на чтение *JA*. N даблов на запись в вектор *result*. 1 дабл = 8 байт. Итого: 8*5N.

Получаем AI для матрично-векторного произведения: $\frac{2N}{8*5N}$, что равно 1/20.

Результаты измерения производительности

Последовательная производительность

Таблица для ЭВМ 1

Статистика ПО/Количество вершин N	**10^4**	**10^5**	**10^6**	**10^7**
Первый этап (генерирование портрета)				
Время выполнения, с	0,038999	0,532	157,832	2042,87
Потребляемая память, МВ	1,2	14	156	1400
Второй этап (построение СЛАУ)				
Время выполнения, с	0,003	0,018999	0,229	1,837
Третий этап (решение СЛАУ)				
Время выполнения, с	0,03	0,168	1,974	18,195
Скалярное произведение векторов				
Общее время выполнения, с	0,001999	0,010999	0,134	1,107
ТВР (AI = $\frac{2N-1}{8*2N}$)	2,624869 GLOPS, 1,051 % TPP	2,624987 GLOPS, 1,051 % TPP	2,624999 GLOPS, 1,051 % TPP	2,624999 GLOPS, 1,051 % TPP
Линейная комбинация				
Общее время выполнения, с	0,003	0,039	0,612998	5,87
ТВР (AI = $\frac{3N}{8*(3N+2)}$)	2,624825 GLOPS, 1,051 % TPP	2,624983 GLOPS, 1,051 % TPP	2,624998 GLOPS, 1,051 % TPP	2,624999 GLOPS, 1,051 % TPP
Матрично-векторное произведение				
Общее время выполнения, с	0,006999	0,062999	0,845001	7,84
ТВР (AI = 1/20)	1,05 GLOPS, 0,42 % TPP	1,05 GLOPS, 0,42 % TPP	1,05 GLOPS, 0,42 % TPP	1,05 GLOPS, 0,42 % TPP

Таблица для ЭВМ 2

Статистика ПО/Количество вершин N	**10^4**	**10^5**	**10^6**	**10^7**
Первый этап (генерирование портрета)				
Время выполнения, с	0,052	0,629	176,355	2249,9
Потребляемая память, МВ	1,2	14	156	1400
Второй этап (построение СЛАУ)				
Время выполнения, с	0,002	0,018	0,281	2,561
Третий этап (решение СЛАУ)				
Время выполнения, с	0,13	0,361	1,869	19,619
Скалярное произведение векторов				
Общее время выполнения, с	0,003999	0,016	0,128	1,098
ТВР (AI = $\frac{2N-1}{8*2N}$)	1,874906 GLOPS, 1,148 % TPP	1,874991 GLOPS, 1,148 % TPP	1,874999 GLOPS, 1,148 % TPP	1,874999 GLOPS, 1,148% TPP
Линейная комбинация				
Общее время выполнения, с	0,002	0,041999	0,496001	5,083
ТВР (AI = $\frac{3N}{8*(3N+2)}$)	1,874875 GLOPS, 1,148 % TPP	1,874988 GLOPS, 1,148 % TPP	1,874999 GLOPS, 1,148 % TPP	1,874999 GLOPS, 1,148% TPP
Матрично-векторное произведение				
Общее время выполнения, с	0,010999	0,085	0,849999	9,409
ТВР (AI = 1/20)	0,75 GLOPS, 0,459 % TPP	0,75 GLOPS, 0,459 % TPP	0,75 GLOPS, 0,459 % TPP	0,75 GLOPS, 0,459 % TPP

Параллельное ускорение

Таблица для ЭВМ 1

Статистика ПО/Количество вершин N	**10^4**	**10^5**	**10^6**	**10^7**
Первый этап (генерирование портрета)				
T = 2, Время выполнения, с	0,036	0,425	96,626	937,766
T = 4. Время выполнения, с	0,042	0,478	56,319	576,27
T = 8. Время выполнения, с	0,054	0,584	42,863	448,981
Второй этап (построение СЛАУ)				
T = 2, Время выполнения, с	0,000999	0,009999	0,112	0,933
T = 4. Время выполнения, с	0,000999	0,005	0,061999	0,49
T = 8. Время выполнения, с	0,000999	0,005	0,061	0,274
Третий этап (решение СЛАУ)				
T = 2, Время выполнения, с	0,033999	0,132	1,703	15,062
T = 4. Время выполнения, с	0,046	0,136	1,487	15,102
T = 8. Время выполнения, с	0,031999	0,177	1,778	15,078
Скалярное произведение векторов				
T = 2, Время выполнения, с	0,001999	0,003	0,100999	1,023
T = 4. Время выполнения, с	0,002	0,003	0,097001	1,045
T = 8. Время выполнения, с	0,003	0,009	0,103999	0,882
Линейная комбинация				
T = 2, Время выполнения, с	0,004999	0,029999	0,573001	5,686
T = 4. Время выполнения, с	0,009999	0,026999	0,498	5,8
T = 8. Время выполнения, с	0,004999	0,04	0,614	5,916
Матрично-векторное произведение				
T = 2, Время выполнения, с	0,006	0,043	0,625999	5,674
T = 4. Время выполнения, с	0,008	0,047	0,549	5,579
T = 8. Время выполнения, с	0,006	0,048	0,657999	5,583

Таблица для ЭВМ 2

Статистика ПО/Количество вершин N	**10^4**	**10^5**	**10^6**	**10^7**
Первый этап (генерирование портрета)				
T = 2, Время выполнения, с	0,049	0,523	106,208	1223,5
T = 4. Время выполнения, с	0,066	0,587	64,399	774,919
T = 8. Время выполнения, с	0,076	0,805	55,768	761,746
Второй этап (построение СЛАУ)				
T = 2, Время выполнения, с	0,001999	0,012	0,141	1,316
T = 4. Время выполнения, с	0,000999	0,006999	0,075	0,832
T = 8. Время выполнения, с	0,000999	0,003999	0,051	0,604
Третий этап (решение СЛАУ)				
T = 2, Время выполнения, с	0,141	0,33	1,558	14,533
T = 4. Время выполнения, с	0,2	0,353	1,608	15,284
T = 8. Время выполнения, с	0,164	0,38	1,919	14,729
Скалярное произведение векторов				
T = 2, Время выполнения, с	0,005	0,013999	0,081001	0,588001
T = 4. Время выполнения, с	0,009	0,019999	0,082999	0,618999
T = 8. Время выполнения, с	0,006999	0,022999	0,103	0,683001
Линейная комбинация				
T = 2, Время выполнения, с	0,009	0,043	0,443999	4,63
T = 4. Время выполнения, с	0,022	0,053999	0,51	5,043
T = 8. Время выполнения, с	0,008999	0,049	0,612	4,714
Матрично-векторное произведение				
T = 2, Время выполнения, с	0,004999	0,061	0,572	5,935
T = 4. Время выполнения, с	0,006999	0,065	0,549	5,777
T = 8. Время выполнения, с	0,005999	0,066999	0,662	5,285

Анализ производительности

ЭВМ 1

Статистика ПО/Количество вершин N	**10^4**	**10^5**	**10^6**	**10^7**
Скалярное произведение векторов				
T = 1, GFLOPS (N)	0,010005	0,018183	0,014925	0,018066
T = 2, GFLOPS (N)	0,010005	0,066666	0,019802	0,01955
T = 4, GFLOPS (N)	0,01	0,066666	0,020618	0,019138
T = 8, GFLOPS (N)	0,006666	0,022222	0,019231	0,022675
Линейная комбинация				
T = 1, GFLOPS (N)	0,01	0,007692	0,004894	0,00511
T = 2, GFLOPS (N)	0,006001	0,01	0,005236	0,005276
T = 4, GFLOPS (N)	0,003	0,011112	0,006024	0,005172
T = 8, GFLOPS (N)	0,006001	0,0075	0,004886	0,00507
Матрично-векторное произведение				
T = 1, GFLOPS (N)	0,002858	0,003175	0,002367	0,00255
T = 2, GFLOPS (N)	0,003333	0,004651	0,003195	0,00352
T = 4, GFLOPS (N)	0,0025	0,004255	0,003643	0,003584
T = 8, GFLOPS (N)	0,003333	0,004167	0,00304	0,003582
3-ий этап (алгоритм решения)				
T = 1, GFLOPS (N)	0,005667	0,010119	0,008612	0,009343
T = 2, GFLOPS (N)	0,005	0,012879	0,009982	0,011286
T = 4, GFLOPS (N)	0,003696	0,0125	0,011432	0,011256
T = 8, GFLOPS (N)	0,005313	0,009605	0,009561	0,011274

ЭВМ 2

Статистика ПО/Количество вершин N	**10^4**	**10^5**	**10^6**	**10^7**
Скалярное произведение векторов				
T = 1, GFLOPS (N)	0,005001	0,0125	0,015625	0,018214
T = 2, GFLOPS (N)	0,004	0,014287	0,024691	0,034013
T = 4, GFLOPS (N)	0,002222	0,01	0,024097	0,03231
T = 8, GFLOPS (N)	0,002857	0,008696	0,019417	0,029282
Линейная комбинация				
T = 1, GFLOPS (N)	0,015	0,007143	0,006048	0,005902
T = 2, GFLOPS (N)	0,003333	0,006977	0,006757	0,006479
T = 4, GFLOPS (N)	0,001364	0,005556	0,005882	0,005948
T = 8, GFLOPS (N)	0,003334	0,006122	0,004902	0,006364
Матрично-векторное произведение				
T = 1, GFLOPS (N)	0,001818	0,002353	0,002353	0,002125
T = 2, GFLOPS (N)	0,004001	0,003279	0,003497	0,003369
T = 4, GFLOPS (N)	0,002858	0,003077	0,003643	0,003462
T = 8, GFLOPS (N)	0,003334	0,002985	0,003021	0,003784
3-ий этап (алгоритм решения)				
T = 1, GFLOPS (N)	0,001308	0,004709	0,009096	0,00866
T = 2, GFLOPS (N)	0,001206	0,005152	0,010911	0,011697
T = 4, GFLOPS (N)	0,00085	0,004816	0,010572	0,011122
T = 8, GFLOPS (N)	0,001037	0,004474	0,008859	0,011541

Анализ полученных результатов

Исходя из данных, приведенных выше видно, что на 2-ух рабочих станциях ТВР операций dot, axpby, spMV находится в диапазоне от 0,5% до 1% ТРР. То есть достигается производительность в районе 1% от пиковой производительности системы. При этом, поскольку обе станции имеют 4 физических ядра и 8 потоков, то наилучшие показатели по скорости выполнения программы достигаются только когда программа запущена с T=8 количеством нитей. Также оба экземпляра оснащены технологией *Intel Turbo Boost*. Именно поэтому в некоторых случаях, когда использовался всего один поток, программа

показывала результаты лучше, чем на нескольких нитях. Также нельзя напрямую сравнивать общие времена, потраченными 3-мя солверами, поскольку каждый раз решение задачи находилось за разное количество итераций, т.е. в каждом из тесте солверы отрабатывали различное количество раз. Рекомендуется сравнивать фактическую производительность солверов. Исходя из приведенных выше сведений о производительности программы от разного количество N и T (количества потоков) можно сделать вывод, что на матрицах больше 10^7 лучше использовать 8 поток, если размер меньше –целесообразнее пробовать 2 или 4 потока. Если же матрица совсем мала, то процессоры, оснащенные технологией Intel Turbo Boost, используя всего 1 поток, могут превосходить в производительности многопоточное выполнение той же самой программы.