Московский государственный университет

Кафедра суперкомпьютеров и квантовой информатики

**Отчет по первому практическому заданию.**

**Однокубитное преобразование вектора-состояния**

**Выполнил: Личманов Дмитрий, студент 323 группы**

**Москва 2020**

**Задание**

1. Реализовать параллельную программу на С++ с использованием OpenMP, которая выполняет однокубитное квантовое преобразование над вектором состояний длины 2n, где n – количество кубитов, по указанному номеру кубита k. Описание однокубитного преобразования дано ниже в разделе методические рекомендации. Для работы с комплексными числами возможно использование стандартной библиотеки шаблонов.

2. Определить максимальное количество кубитов, для которых возможна работа программы на системе Polus. Выполнить теоретический расчет и проверить его экспериментально.

3. Протестировать программу на системе Polus. В качестве теста использовать преобразование Адамара по номеру кубита:

a) который соответствует номеру в списке группы плюс 1.

b) 1

c) n

Начальное состояние вектора должно генерироваться случайным образом. Заполнить таблицу и построить график зависимости ускорения параллельной программы от числа процессоров для каждого из случаев a)-c):

**Описание алгоритма**

Однокубитная операция задается двумя параметрами: комплексной матрицей размера 2*х*2 и числом от 1 до *n* (данный параметр обозначает номер кубита, по которому проводится операция).

Итак, дана комплексная матрица: 

и *k* - номер индекса от 1 до *n* (номер кубита).

Такая операция преобразует вектор  в , где все элементов нового вектора вычисляются по следующей формуле:

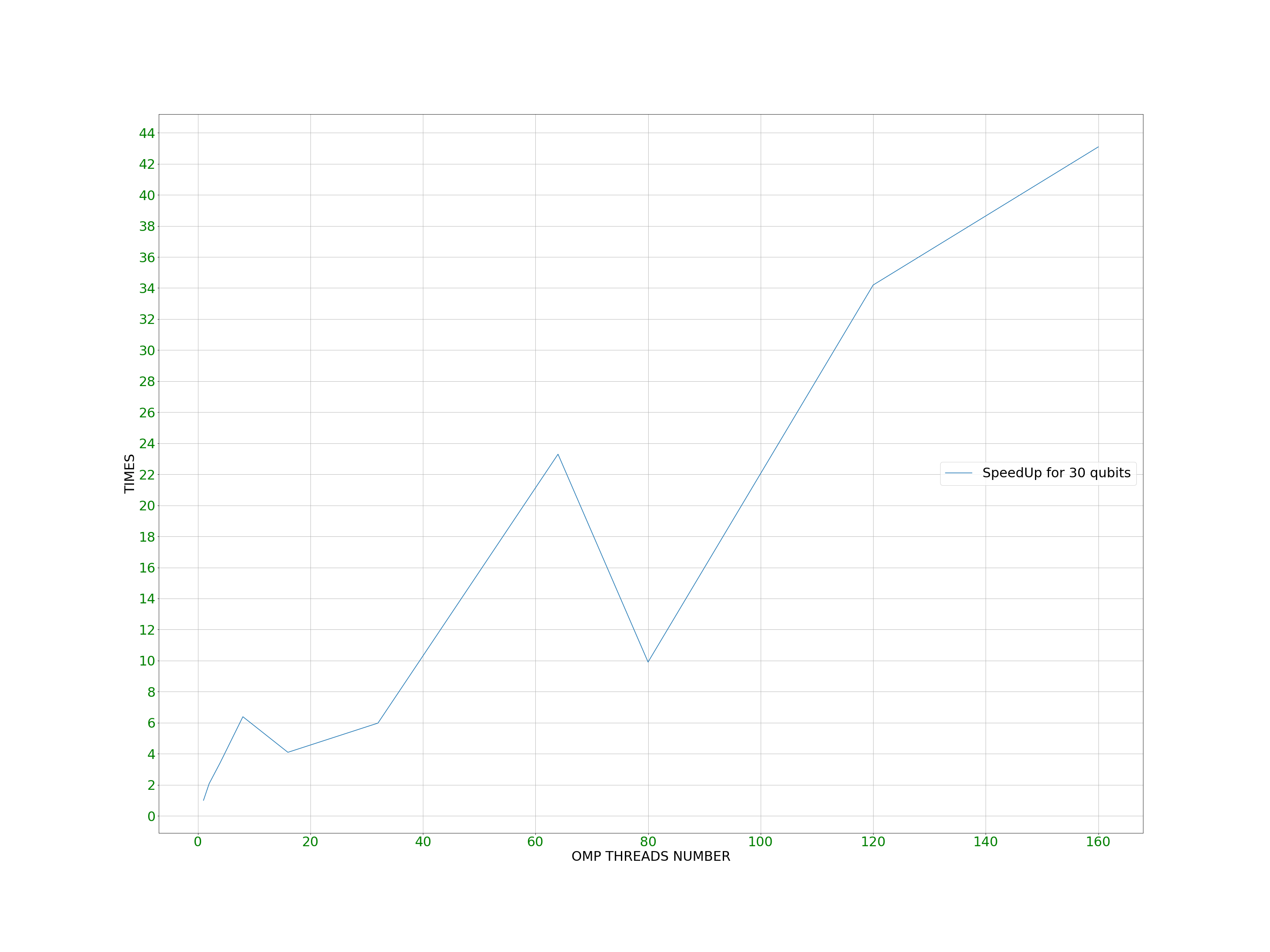


|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Количество кубитов** | **Количество процессоров** | **Время работы программы(сек)** | | | **Ускорение** | | |
| **k =1** | **k = 3** | **k = n** | **k = 1** | **k = 3** | **k = n** |
| **20** | 1 | 0.092988 | 0.093377 | 0.092983 | 1 | 1 | 1 |
| 2 | 0.047803 | 0.046877 | 0.046870 | 1,95744681 | 2,02173913 | 1,95744681 |
| 4 | 0.033854 | 0.026129 | 0.024177 | 2,78787879 | 3,57692308 | 3,40740741 |
| 8 | 0.019135 | 0.023488 | 0.017232 | 4,84210526 | 4,17391304 | 4,84210526 |
| 160 | 0.007175 | 0.007114 | 0.009632 | 13,084507 | 13,1408451 | 9,67708333 |
| **24** | 1 | 1.495206 | 1.568541 | 1.494971 | 1 | 1 | 1 |
| 2 | 1.238219 | 0.752969 | 0.783435 | 1,21138211 | 1,68 | 1,91025641 |
| 4 | 0.387888 | 0.395535 | 0.439796 | 3,92105263 | 3,80487805 | 3,46511628 |
| 8 | 0.394153 | 0.605384 | 0.657770 | 3,82051282 | 2,55737705 | 2,29230769 |
| 160 | 0.058612 | 0.126544 | 0.061346 | 28,2641509 | 12,4444444 | 24,4918033 |
| **28** | 1 | 24.901079 | 24.637257 | 24.216557 | 1 | 1 | 1 |
| 2 | 12.183924 | 12.397169 | 12.461979 | 2,05785124 | 1,98387097 | 1,9516129 |
| 4 | 6.200570 | 6.263630 | 6.191769 | 4,01612903 | 3,9047619 | 3,96721311 |
| 8 | 3.507420 | 3.288214 | 3.486420 | 7,11428571 | 7,6875 | 7,11764706 |
| 160 | 0.692365 | 1.00408 | 0.714187 | 36,0869565 | 24,63 | 34,0985915 |
| **максимальное** | 1 | 105.996422 | 95.797493 | 95.897324 | 1 | 1 | 1 |
| 2 | 51.216545 | 49.733566 | 49.900890 | 2,05882353 | 1,93877551 | 1,93877551 |
| 4 | 30.745071 | 25.223854 | 24.811956 | 3,38709677 | 4,13043478 | 3,95833333 |
| 8 | 16.689852 | 13.178022 | 12.992264 | 6,5625 | 7,30769231 | 7,91666667 |
| 160 | 5.221394 | 7.961027 | 5.972296 | 20,3653846 | 12,1139241 | 16,2372881 |

Вычисления программы проводились на системе IBM Polus

Для оценки времени выполнения однокубитного преобразования использовалась функция gettimeofday(), возвращающая системное время в микросекундах

Ускорение выполнения многопоточной программы = отношение времени выполнения однопоточного преобразования к времени выполнения многопоточного однокубитного преобразования

**Выводы**

Естественно, при росте числа кубитов наблюдается экспоненциальный рост времени выполнения преобразования. Однако, номер преобразуемого кубита не сильно повлиял на время выполнения преобразования. Что касается полученного графика, для 80 потоков наблюдается небольшое отклонение от общей регрессии ускорения, однако это связано сугубо с оптимизацией накладных расходов при определенном числе потоков. Распараллеливание эффективно и ускорение сравнимо с количеством вычислительных потоков. Максиамльный размер вектора, при котором не возникало ошибки выделения памяти bad\_alloc - элементов.