# Постановка задачи

Завдання: На мові С/C++ напишіть програму яка буде перемножувати квадратну матрицю розміром NxN на вектор розмірність N. Величину N виберіть таким чином, щоб максимально використати всю можливу оперативну пам’ять. Обчисліть затрати часу на дану дію. Перемножте також вектор на матрицю. Порівняйте затрачений час в обох випадках. Зробіть висновки. Запропонуйте зміну алгоритму таким чином, щоб затрачений час був приблизно однаковий.

Замерить время перемножения матрицы на вектор, вектора на матирицу, а так же время оптимизированного способа перемножения вектора на матрцу. Замерить время для размеров матрицы от N=1000 до максимального размера матрицы который удалось разместить в оперативной памяти с шагом 1000.

Произвести перемножения и засечь время для разных типов данных: int, float, double.

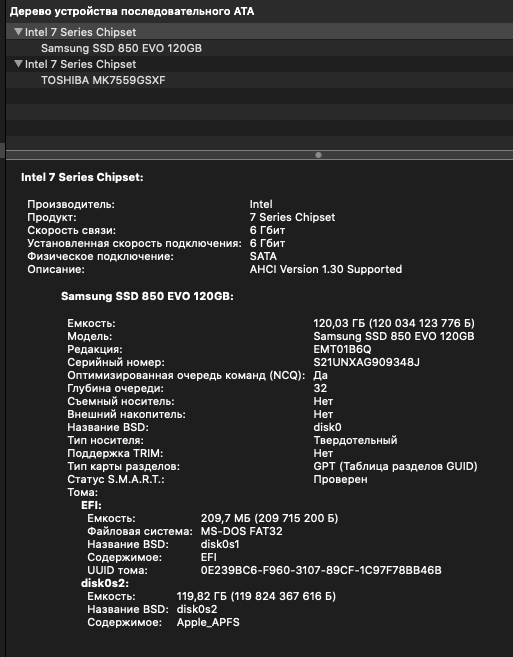
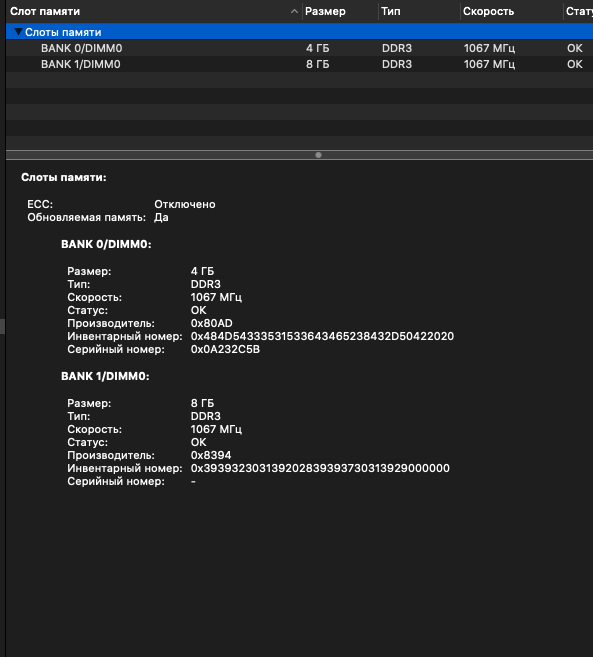
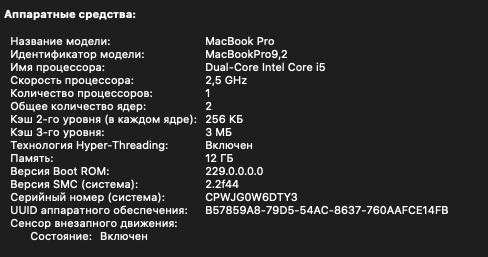
Создать таблицы с куда занести результаты засечек времени.

Создать графики на которых отобразить данные с таблиц.

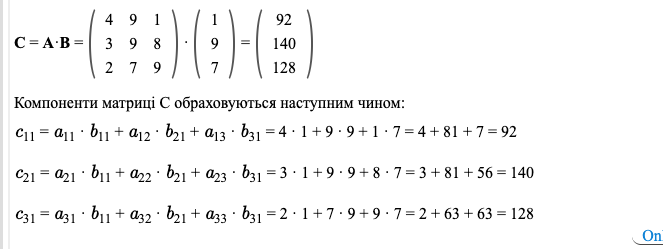
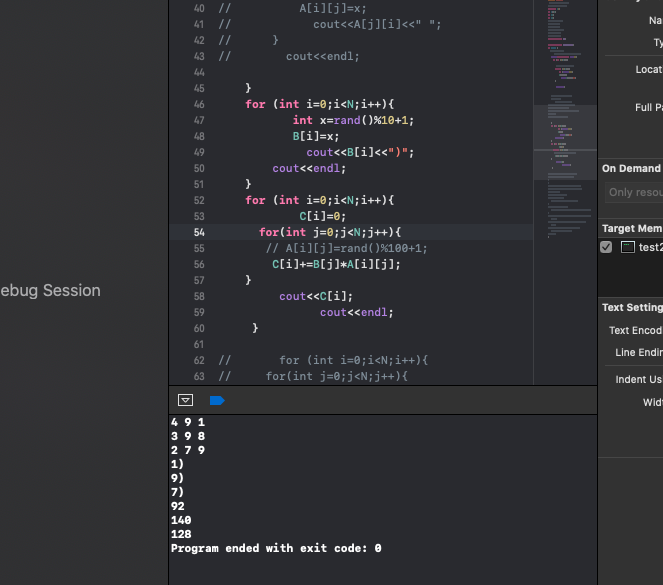
Сравнить время перемножения разных типов данных и разных способов перемножения

# Характеристики компьютера

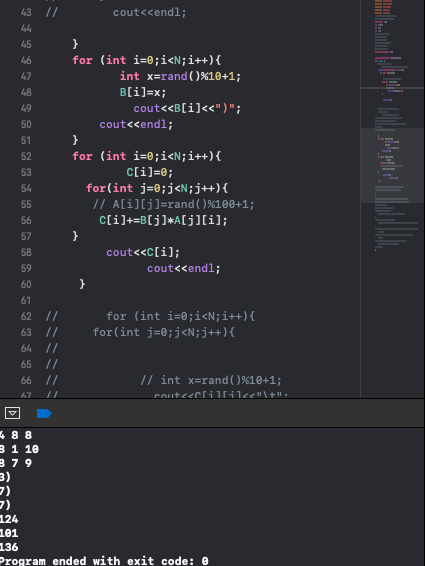
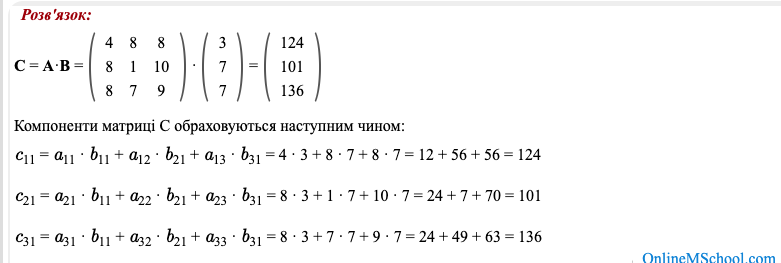




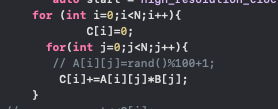
Проверка на правильность умножения матрицы

Быстрое умножение  
  
  
  


Медленное умножение



Умножение матрицы на вектор



Замеры скорости выполнения

Примечание\*   
Все значения представлены в секундах

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | int | | | float | | | double | | |
| **1000** | fast | slow | reverse | fast | slow | reverse | fast | slow | reverse |
| **2000** | **0,02101** | **0,10220** | **0,02672** | **0,02344** | **0,08912** | **0,02253** | **0,04476** | **0,14686** | **0,03725** |
| **3000** | **0,05868** | **0,24962** | **0,05097** | **0,04668** | **0,22528** | **0,04869** | **0,09338** | **0,22073** | **0,09432** |
| **4000** | **0,12052** | **0,44409** | **0,08324** | **0,09276** | **0,43342** | **0,10502** | **0,12878** | **0,46589** | **0,15311** |
| **5000** | **0,14750** | **0,66868** | **0,14077** | **0,13472** | **0,72884** | **0,13911** | **0,26315** | **0,86358** | **0,29350** |
| **6000** | **0,21046** | **0,98120** | **0,20255** | **0,20268** | **0,98430** | **0,19072** | **0,37980** | **1,20691** | **0,31833** |
| **7000** | **0,25781** | **1,37773** | **0,27916** | **0,26588** | **1,38174** | **0,26635** | **0,46861** | **1,74062** | **0,48961** |
| **8000** | **0,34422** | **1,91763** | **0,42196** | **0,34547** | **1,87297** | **0,34769** | **0,65095** | **2,30203** | **0,59024** |
| **9000** | **0,44455** | **2,43586** | **0,51544** | **0,43650** | **2,44543** | **0,44911** | **0,68594** | **2,98001** | **0,78194** |
| **10000** | **0,53519** | **3,00981** | **0,57488** | **0,53255** | **2,57854** | **0,54396** | **0,90753** | **3,35632** | **0,92720** |
| **11000** | **0,67294** | **3,55082** | **0,75663** | **0,64320** | **3,32340** | **0,62695** | **1,08366** | **4,42172** | **1,10633** |
| **12000** | **0,78423** | **4,24378** | **0,95720** | **0,84421** | **3,79283** | **0,75468** | **1,50021** | **5,95631** | **1,59137** |
| **13000** | **0,91512** | **5,03800** | **1,34744** | **0,96285** | **5,10266** | **0,89920** | **1,77124** | **7,32880** | **1,88443** |
| **14000** | **1,35159** | **6,20687** | **1,26304** | **1,16567** | **5,78800** | **1,02599** | **2,05997** | **10,21990** | **2,07214** |
| **15000** | **1,38879** | **7,53655** | **1,26743** | **1,26583** | **8,01486** | **1,19712** | **2,32962** | **12,69540** | **2,37637** |
| **16000** | **1,68630** | **9,42419** | **2,02392** | **1,99110** | **9,75940** | **1,45928** | **2,84983** | **14,14980** | **2,70190** |
| **17000** | **1,84599** | **10,13100** | **2,10065** | **1,84305** | **10,38910** | **2,01171** | **3,05738** | **12,99630** | **3,08825** |
| **18000** | **2,11073** | **13,45680** | **2,42280** | **2,16270** | **12,21930** | **2,13828** | **3,41524** | **14,98570** | **3,42756** |
| **19000** | **2,48048** | **14,29280** | **2,49840** | **2,49311** | **14,51720** | **2,29162** | **3,95120** | **16,34900** | **3,78517** |
| **20000** | **2,53497** | **16,61880** | **3,00731** | **2,65544** | **15,83480** | **2,56061** | **4,21639** | **18,91770** | **4,31847** |
| **21000** | **2,89084** | **18,67860** | **3,03144** | **3,05341** | **18,90690** | **2,99120** | **4,66178** | **20,33410** | **4,65899** |
| **22000** | **3,17869** | **21,19270** | **3,17976** | **3,19849** | **20,16290** | **3,11039** | **5,20398** | **22,46690** | **5,15120** |
| **23000** | **3,55089** | **23,80930** | **3,49991** | **3,70019** | **23,66420** | **4,07618** | **5,53504** | **25,24400** | **5,81056** |
| **24000** | **3,76517** | **28,01980** | **3,86604** | **3,97594** | **26,22890** | **5,29374** | **6,02843** | **28,44020** | **6,24049** |
| **25000** | **4,08506** | **28,42440** | **4,35426** | **4,34133** | **30,33100** | **5,63924** | **6,57907** | **31,00520** | **6,68605** |
| **26000** | **4,46949** | **31,29650** | **4,64430** | **4,59111** | **49,98940** | **5,45285** | **7,01741** | **33,91490** | **7,28687** |
| **27000** | **4,73103** | **34,04160** | **5,39687** | **4,96162** | **52,30500** | **5,30320** | **7,68316** | **45,19120** | **7,76109** |
| **28000** | **5,23205** | **37,38630** | **5,31256** | **5,37871** | **43,24560** | **5,66781** | **8,21506** | **45,88440** | **8,38594** |
| **29000** | **5,78149** | **45,96390** | **5,56285** | **6,04268** | **42,40080** | **6,40022** | **8,86609** | **56,47310** | **8,82450** |
| **30000** | **6,45230** | **47,81530** | **6,07503** | **6,21340** | **50,18720** | **6,11478** | **9,43051** | **59,18650** | **9,66249** |

График быстрого медленного и обратного умножения для int

График быстрого медленного и обратного умножения для float

График быстрого медленного и обратного умножения для double

Можно заметить, что при быстром умножении для int и float значения очень похожи, но при других умножениях они отличаются.

Также на графиках заметен всплеск для float на 26000 после чего опять снижение.

При умножении матриц и вектора типа double проходит больше времени т.к. Он занимает больше памяти чем типы int float

Не оптимизирование умножение действительно медленное и занимает в 8-10 больше времени чем быстрое и оптимизированное умножение при всех испытуемых типах данных

# 

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include <chrono>

//const int N=250000;

//const int N= 1000;

//float A[N][N];

//float B[N];

//float C[N];

//double A[N][N];

//double B[N];

//double C[N];

**using** **namespace** std;

**using** **namespace** std::chrono;

**int** main() {

**int** N=1000;

// cout<<endl;

//cout<<"trans"<<endl;

// B[i]=x;

//cout<<B[i]<<")";

// for(int j=0;j<N;j++){

// A[i][j]=x;

// cout<<A[j][i]<<" ";

// }

// cout<<endl;

**do**{

**double** \*\* A= **new** **double**\*[N];

**double** \*B= **new** **double** [N];

**double** \*C=**new** **double** [N];

**for**( **int** i=0; i<N; i++){

A[i]=**new** **double**[N];

}

**for** (**int** i=0;i<N;i++){

**double** x=rand()%10+1;

B[i]=x;

// cout<<B[i]<<")";

// cout<<endl;

}

**auto** start = high\_resolution\_clock::now();

**for** (**int** i=0;i<N;i++){

C[i]=0;

**for**(**int** j=0;j<N;j++){

// A[i][j]=rand()%100+1;

C[i]+=A[i][j]\*B[j];

}

// cout<<C[i];

// cout<<endl;

}

// for (int i=0;i<N;i++){

// for(int j=0;j<N;j++){

//

//

// // int x=rand()%10+1;

// cout<<C[i][j]<<"\t";

// }

// cout<<endl;

**auto** stop = high\_resolution\_clock::now();

//

std::chrono::duration<**double**> duration = stop-start;

// cout<<N<<" "<<endl;

cout << duration.count() <<"" << endl;

// }

N+=1000;

}**while** (N<=30000);

}