Politechnika Opolska
Wydział Elektrotechniki, Automatyki i Informatyki
Instytut Informatyki

Programowanie współbieżne i rozproszone
Informatyka, studia stacjonarne II stopnia
Laboratorium

Sprawozdanie

Rozwiązywanie układów równań liniowych metodą eliminacja Gaussa

Prowadzący:
dr hab. inż. Jan Sadecki, prof. PO

Ćwiczenie zrealizował:
Marcin Kuchnia
nr indeksu 93210
grupa L1

Opole, maj 2020

Metody obliczeniowe

W ramach ćwiczenia zmierzono czasy rozwiązywania układów równań liniowych metodą eliminacji Gaussa. Jest to algorytm, który nie daje się w prosty i oczywisty sposób zrównoleglić, ponieważ niektóre obliczenia opierają się na wynikach wcześniejszych działań. Przeprowadzono obliczenia na czterech rozmiarach macierzy: 1600, 2400, 3200 oraz 4000 z wykorzystaniem od 1 do 12 wątków

Funkcje losujące, generujące i wyświetlające macierze

Algorytm realizowany był na wygenerowanych losowo macierzach. Funkcja *fRand()* zwraca wartość zmiennoprzecinkową o wartości od -1000 do 1000. Podobna funkcja *intRand()* zwraca liczbę całkowitą od -3 do 3 i wykorzystywana jest tylko w celach testowych, weryfikujących poprawność działania algorytmu. Funkcja *generateRandomMatrix()* pobiera jako parametry rozmiar macierzy oraz wskaźnik do pierwszego elementu. Korzystając z odpowiedniej funkcji losującej wypełnia ona macierz losowymi wartościami. Funkcja *printMatrix()* umożliwia wyświetlenie zadanej macierzy w celu sprawdzenia jej zawartości. Daje to możliwość weryfikacji poprawności obliczeń.

Implementacja funkcji

```
float fRand()
{
    float fMin = -1000;
    float fMax = 1000;
    float f = (float)rand() / RAND_MAX;
    return fMin + f * (fMax - fMin);
}

int intRand()
{
    int fMin = -3;
    int fMax = 3;
    int f = rand()%(fMax-fMin)+fMin;
    return f;
}

void generateRandomMatrix(int n, float ** Matrix)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n + 1; j++)
        {
            Matrix[i][j] = fRand();
            if (TEST) Matrix[i][j] = intRand();
        }
    }
}

void printMatrix(int n, float** Matrix)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n + 1; j++)
        {
            cout << setw(10) << left << Matrix[i][j];
        }
        cout << endl;
    }
}
```

Sekwencyjny algorytm eliminacji Gaussa

Funkcja ta jest stosowana do wygenerowania poprawnej macierzy, możliwej do przekształcenia przez funkcję wielowątkową. Funkcja ta przeprowadza eliminację Gaussa bez wykorzystania obliczeń równoległych. W pierwotnej wersji po napotkaniu na przekątnej elementu o wartości 0 można zastosować zamianę wierszy. Możliwość ta została jednak pominięta, ponieważ algorytm wielowątkowy nie jest w stanie zrealizować takiego rozwiązania. Po napotkaniu liczby 0 funkcja jest po prostu przerywana.

Algorytm w kolejnych iteracjach pobiera element znajdujący się na przekątnej i dzieli przez niego wszystkie elementy znajdujące się w wierszu za nim. Tak obliczony wiersz ma jest wykorzystywany do obliczenia kolejnych wierszy. Element znajdujący się na przekątnej przyjmuje wartość 1.

W kolejnych wierszach wszystkie pozostałe elementy (jeszcze nie wyeliminowane) są pomniejszane o iloczyn pierwszego elementu z wiersza oraz odpowiadającego im elementu z wiersza obliczonego na początku iteracji. W efekcie pierwsze obliczane elementy w wierszach przyjmują wartość 0 – zostają wyeliminowane, i nie są już stosowane w dalszych obliczeniach.

Efektem działania algorytmu powinna być macierz schodkowa o wartościach na przekątnej równych 1.

Implementacja alogorytmu

```
void GaussElimination(int n, float ** AB)
{
    for (int k = 0; k < n; k++)
    {
        //jeśli pierwszy rozpatrywany element na przekątnej jest równy 0, to zamieniaj z kolejnymi
wierszami (o ile istnieją) do skutku lub do końca wierszy
        //int nextRow = k + 1;
        //while (AB[k][k] == 0 && nextRow < n)
        //{
        //    for (int j = 0; j < n; j++)
        //    {
        //        swap(AB[k][j], AB[nextRow][j]);
        //    }
        //    nextRow += 1;
        //}
        if (AB[k][k] == 0)
        {
            break;
        }
        //OBLICZENIA
        for (int j = k + 1; j < n + 1; j++)
        {
            AB[k][j] = AB[k][j] / AB[k][k];
        }
        AB[k][k] = 1;
        //ELIMINACJA
        for (int i = k + 1; i < n; i++)
        {
            for (int j = k + 1; j < n + 1; j++)
                AB[i][j] = AB[i][j] - AB[i][k] * AB[k][j];
            AB[i][k] = 0;
        }
    }
}
```

Równoległy algorytm eliminacji Gaussa

Konstrukcja algorytmu równoległego jest niemal identyczna z algorytmem sekwencyjnym. Dla każdej iteracji wykonywane są dwa etapy: obliczanie pierwszego wiersza oraz eliminacja kolejnych wierszy. To właśnie krok eliminacji może zostać zrównoleglony. Wiersze mogą być przydzielane do wierszy blokowo (kilka kolejnych wierszy) lub cyklicznie (każdy wątek po zakończeniu poprzedniego zadania otrzymuje następny, nieobliczany jeszcze wiersz). Różnicę w programie stanowi występowanie dyrektywy *schedule(dynamic, 1)*. W trakcie badania czasów nie zauważono znaczących różnić pomiędzy tymi wersjami, zdecydowano się więc dogłębnie przeanalizować wersję cykliczną.

Podjęto również bezowocne próby stworzenia wersji potokowej, która szerzej została opisana na końcu sprawozdania. Teoretycznie możliwe jest także zrównoleglenie obliczeń pierwszego wiersza, jednak narzut związany z wykorzystaniem wielu wątków tylko pogarszał czasy wykonania, rozwiązanie to zostało więc zarzucone.

Implementacja

```
void multiGaussElimination(int n, float** AB)
{
    for (int k = 0; k < n; k++)
    {
        for (int i = k + 1; i < n + 1; i++)
        {
            AB[k][i] = AB[k][i] / AB[k][k];
        }
        AB[k][k] = 1;

        #pragma omp parallel for schedule(dynamic, 1) firstprivate(k, n) shared(AB)
        for (int i = k + 1; i < n; i++)
        {
            for (int j = k + 1; j < n + 1; j++)
                AB[i][j] = AB[i][j] - AB[i][k] * AB[k][j];
            AB[i][k] = 0;
        }
    }
}
```

Rozwiązanie układu równań

Przeprowadzenie eliminacji Gaussa nie jest jeszcze etapem finalnym rozwiązywania układu równań liniowych. Do tego celu została stworzona funkcja *GaussSolve()*. Jej parametrami są rozmiar badanego układu *n*, macierz rozszerzona układu *AB* oraz macierz wynikowa *X*. Najpierw na macierzy układu realizowany jest algorytm eliminacji. Jeśli program znajduje się w fazie testowej otrzymana macierz jest wyświetlana w celu weryfikacji wyników. Następnie następuje podstawianie wsteczne: element ostatni daje się wyliczyć wprost. Element przedostatni daje się wyliczyć na podstawie elementu ostatniego, element trzeci od końca na podstawie przedostatniego, itd. Otrzymane wyniki zapisywane są do macierzy wynikowej *X*.

Implementacja

```
void GaussSolve(int n, float** AB, float* X)
{
    multiGaussElimination(n, AB);
    if (TEST) printMatrix(n, AB);
    for (int i = n - 1; i >= 0; i--)
    {
        float s = AB[i][n];
        for (int j = n - 1; j > i; j--)
        {
            s -= AB[i][j] * X[j];
        }
        X[i] = s / AB[i][i];
    }
}
```

Funkcja testująca

W celu pomiaru czasu obliczeń zaprojektowano funkcję testującą *multipleTestExecution*, która oprócz parametrów dla funkcji realizujących algorytm przyjmuje także ilość testów do wykonania. Kolejne pomiary zapisywane są w wektorze *executeTimes*. Jednostką pomiaru czasu jest milisekunda. Mierzony jest tylko czas rozwiązywania układu. W czasie obliczania wyświetlany jest progres, w postaci *wykonanoTestów/liczbaTestów*. Po zakończeniu testów wyświetlana oraz zapisywana do pliku (za pomocą strumienia *fs*) jest liczba wątków oraz pojedyncze czasy posortowane rosnąco.

Jeśli program uruchomiony jest w trybie testu to nie jest wyświetlany progres obliczeń oraz zmierzony czas, tylko wyświetlana jest tablica z obliczonymi wynikami, co pozwala zweryfikować poprawność działania.

Implementacja

```
void multipleTestExecution(int n, float** AB, float* X, int maxTests)
{
    //badanie czasu obliczania
    vector < long long > executeTimes;
    for (int i = 1; i <= maxTests; i++)
    {
        if (!TEST) cout << '\r' << i << '/' << maxTests;
        //kopia lokalna wygenerowanej macierzy
        float** localMatrix;
        localMatrix = new float* [n];
        for (int j = 0; j < n; j++) localMatrix[j] = new float[n + 1];
        for (int j = 0; j < n; j++)
        {
            for (int k = 0; k < n + 1; k++)
            {
                localMatrix[j][k] = AB[j][k];
            }
        }
        //pomiar czasu obliczania
        chrono::steady_clock::time_point start = chrono::steady_clock::now();
        GaussSolve(n, localMatrix, X);
        chrono::steady_clock::time_point end = chrono::steady_clock::now();
        //END pomiar czasu obliczania
        for (int j = 0; j < n; j++) delete[] localMatrix[j];
        delete[] localMatrix;
        executeTimes.push_back(chrono::duration_cast<chrono::milliseconds>(end - start).count());
        //pokaż wyniki
        if (TEST) for (int j = 0; j < n; j++) cout << "X" << j << ": " << X[j] << endl;
    }
    if (!TEST) cout << "\r                      \r";
    //wyświetlanie posortowanych czasów obliczania
    sort(executeTimes.begin(), executeTimes.end());
    if (!TEST) cout << setw(10) << left << omp_get_max_threads(); fs << setw(10) << left <<
omp_get_max_threads();
    if (!TEST) for (const auto& i : executeTimes)
    {
        cout << setw(10) << left << i; fs << setw(10) << left << i;
    }
    cout << endl; fs << endl;
    //END wyświetlanie posortowanych czasów obliczania

    //END badanie czasu obliczania
}
```

Program główny

Funkcja *main()* realizująca główne zadania korzysta z zaimplementowanych funkcji. Na początku ustawiany jest punkt czasu pozwalający mierzyć czas całego badania. Następnie ustawiane są opcje językowe oraz ustalane jest ziarno generatora liczb pseudolosowych na podstawie czasu. Dalej ustawiane są parametry programu: rozmiary macierzy, liczba testów i wykorzystanych wątków. Otwierany jest także plik do którego zapisywane są wyniki.

Dla każdego rozmiaru tworzone i generowane są nowe macierze. W trybie testowym dodatkowo wygenerowana macierz jest wyświetlana. Dla wszystkich liczb wątków uruchamiana jest funkcja badająca czasy. Da trybu testowego uruchamiany jest tylko raz, z czterema wątkami. Po zakończeniu badania dynamiczne tablice są usuwane. Na zakończenie programu wyświetlany jest sumaryczny czas jego działania.

Implementacja

```
int main()
{
    chrono::steady_clock::time_point start = chrono::steady_clock::now();
    setlocale(LC_CTYPE, "Polish");
    srand(time(NULL));
    int matrixSizes[] = { 1600, 2400, 3200, 4000 };
    if (TEST) { matrixSizes[0] = 3; matrixSizes[1] = 4; matrixSizes[2] = 5; matrixSizes[3] = 6; }
    int maxTests = 100;
    if (TEST) maxTests = 1;
    int maxThreads = 12;
    if (TEST) maxThreads = 1;
    fs.open("out.txt", fstream::in | fstream::out | fstream::trunc);

    //pętla dla wszystkich rozmiarów macierzy
    for (int i = 0; i < 4 ; i++)
    {
        int n = matrixSizes[i];
        cout << n << endl; fs << n << endl;
        //dynamiczna macierz rozszerzona
        float ** AB;
        AB = new float* [n];
        for (int j = 0; j < n; j++) AB[j] = new float[n + 1];
        //dynamiczna macierz niewiadomych
        float * X;
        X = new float[n];

        generateRandomCorrectMatrix(n, AB);
        if (TEST) printMatrix(n, AB);
        //pętla dla wszyskich liczb wątków
        for (int j = 1; j <= maxThreads;j++)
        {
            omp_set_num_threads(j);
            if (TEST) omp_set_num_threads(4);
            multipleTestExecution(n, AB, X, maxTests);
        }
        //usuwanie macierzy dynamicznych
        for (int j = 0; j < n; j++) delete[] AB[j];
        delete[] AB;
        delete[] X;
    }

    fs.close();
    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    int sec = chrono::duration_cast<chrono::seconds>(end - start).count();
    int min = chrono::duration_cast<chrono::minutes>(end - start).count();
    int milisec = chrono::duration_cast<chrono::milliseconds>(end - start).count();
    cout << "Zakończono: " << min << "m " << sec << "s " << milisec << "ms " << endl;
    getchar();
```

Pomiary

Program został stworzony i skompilowany w programie Visual Studio 2019. Uruchomiony został na urządzeniu o parametrach określonych w tabeli:

LENOVO	Lenovo B50-80 Procesor Intel(R) Core(TM) i3-5020U CPU @ 2.20GHz, 2200 MHz, Rdzenie: 2, Procesory logiczne: 4 Nazwa systemu operacyjnego Microsoft Windows 10 Education

Wszystkie otrzymane wartości zestawiono w tabelach na końcu sprawozdania.

Wśród otrzymanych wyników zdarza się, że najdłuższe czasy wykonania odstają od średniej wartości. W większości nie przekraczają one jednak dwukrotności średniej , a ich wpływ na wartość średniej nie jest znaczący. Odrzucenie 20% najdłuższych czasów poprawia jednak stabilność otrzymanych wyników, są one bardziej spójne. Ogólnie obserwowany trend wskazuje, że najbardziej odstające wyniki zdarzają się dla najmniejszego rozmiaru macierzy.

Analiza wyników

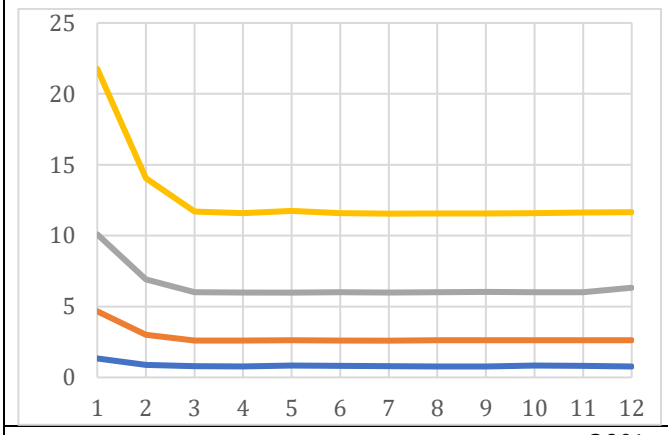
Średni czas wykonania
Oś pozioma: liczba wątków 1-12
Oś pionowa: średni czas w s

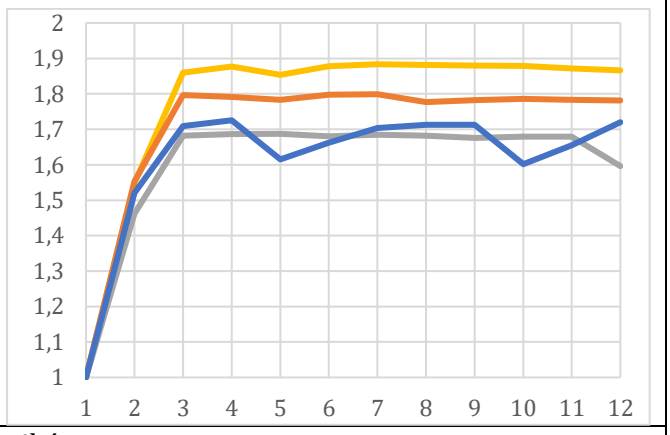
Rozmiar macierzy
4000
3200
2400
1600

Przyspieszenie
Oś pozioma: liczba wątków 1-12
Oś pionowa: przyspieszenie

100% wyników

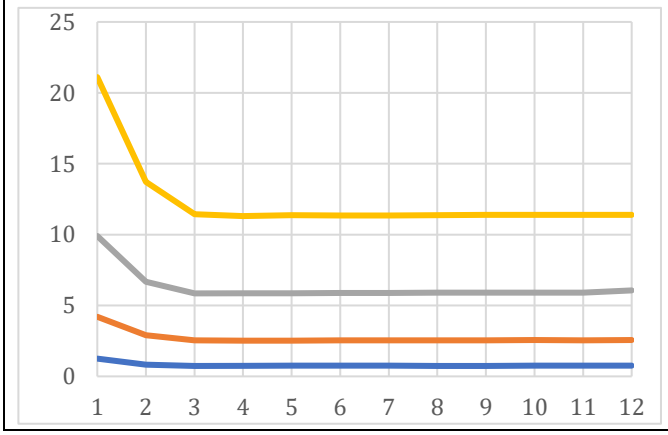
Wątków	1	2	3	4	5	6	7	8	9	10	11	12
Średni czas	21,76	14,05	11,70	11,59	11,74	11,59	11,55	11,57	11,57	11,58	11,62	11,66
Przyspieszenie	1,00	1,55	1,86	1,88	1,85	1,88	1,88	1,88	1,88	1,88	1,87	1,87
Średni czas	10,09	6,90	6,00	5,98	5,98	6,01	5,99	6,00	6,02	6,01	6,01	6,32
Przyspieszenie	1,00	1,46	1,68	1,69	1,69	1,68	1,68	1,68	1,68	1,68	1,68	1,60
Średni czas	4,66	3,00	2,60	2,60	2,61	2,59	2,59	2,62	2,62	2,61	2,62	2,62
Przyspieszenie	1,00	1,55	1,80	1,79	1,78	1,80	1,80	1,78	1,78	1,79	1,78	1,78
Średni czas	1,34	0,88	0,78	0,78	0,83	0,80	0,79	0,78	0,78	0,84	0,81	0,78
Przyspieszenie	1,00	1,52	1,71	1,73	1,61	1,66	1,70	1,71	1,71	1,60	1,65	1,72

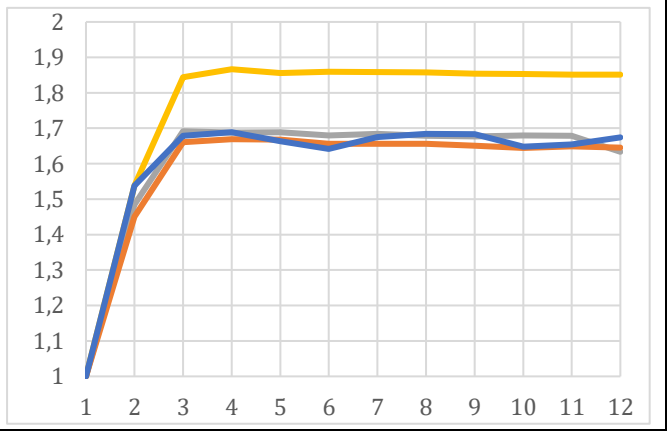




80% wyników

Wątków	1	2	3	4	5	6	7	8	9	10	11	12
Średni czas	21,10	13,72	11,45	11,31	11,37	11,35	11,36	11,36	11,38	11,39	11,40	11,40
Przyspieszenie	1,00	1,54	1,84	1,87	1,86	1,86	1,86	1,86	1,85	1,85	1,85	1,85
Średni czas	9,90	6,67	5,85	5,87	5,86	5,89	5,88	5,90	5,90	5,90	5,90	6,06
Przyspieszenie	1,00	1,48	1,69	1,69	1,69	1,68	1,68	1,68	1,68	1,68	1,68	1,63
Średni czas	4,20	2,90	2,53	2,52	2,52	2,54	2,54	2,54	2,55	2,56	2,55	2,55
Przyspieszenie	1,00	1,45	1,66	1,67	1,67	1,66	1,66	1,66	1,65	1,64	1,65	1,65
Średni czas	1,25	0,82	0,75	0,74	0,75	0,76	0,75	0,74	0,75	0,76	0,76	0,75
Przyspieszenie	1,00	1,54	1,68	1,69	1,66	1,64	1,68	1,68	1,68	1,65	1,65	1,67





Na wykresach można zaobserwować, że algorytm realizowany z wykorzystaniem wielu wątków może być wykonywany szybciej niż używając pojedynczego procesu. Największe różnice czasu obserwowalne są dla obliczeń na największej ilości danych. Rozpatrując jednak przyspieszenie jako poprawę w stosunku do czasu wykonywania procesu jednowątkowego, to jest ono bardzo zbliżone dla wszystkich rozmiarów. W wielu miejscach wykresy reprezentujące przyspieszenie pokrywają się nawet dla różnych wielkości rozpatrywanego problemu. Najbardziej znaczący wzrost przyspieszenia zachodzi dla liczby wątków od 1 do 3. Poprawa dla 4 wątków jest wyraźnie mniejsza, a dla obliczeń na macierzy wielkości 2400 następuje wręcz niewielkie pogorszenie wyników.

Wyraźny regres przyspieszenia obserwowany jest dla najmniejszej macierzy przy przejściu z 4 na 5 wątków oraz z 9 na 10 wątków. Może być to powiązane z wykorzystaniem do obliczeń procesora **czterowątkowego**. Nadmiarowe wątki są zmuszone bezczynnie oczekiwać na wolny procesor logiczny. Większa liczba danych oraz wątków powoduje jednak, że regres ten jest mniejszy. Maksymalne wartości przyspieszenia są zbliżone do wyników uzyskanych przy wykorzystaniu trzech wątków.

Odrzucenie 20% najdłuższych czasów wykonania pozwala uzyskać bardziej spójne wyniki. Uzyskane wykresy cechują się mniejszą liczbą wahań, co może świadczyć że wartości najdłuższe mogły zostać uzyskane w wyniku zakłóceń działania programu. Pokazuje to wrażliwość mniej skomplikowanych obliczeń na zakłócenia pracy.

Otrzymane wyniki mogłyby być bardziej reprezentatywne, gdyby program został uruchomiony w lepiej przygotowanym środowisku. Odciążenie systemu poprzez wyłączenie wszystkich niepotrzebnych programów i funkcji pozostawiłoby procesor niemal w całości do dyspozycji badanego zadania. Tak otrzymane wartości mogłyby być bardziej zbliżone do teoretycznych modeli.

Algorytm asynchroniczny

Stworzona funkcja próbująca realizować algorytm asynchroniczny wciąż nie uzyskała ostatecznego kształtu. W kodzie pozostawiono więc komentarze i fragmenty wykorzystywane w trakcie testów. Przykłady odwołują się do macierzy o 5x5, czyli po rozszerzeniu 5x6.

Algorytm w wersji asynchronicznej pozwala teoretycznie uzyskać lepsze wykorzystanie współbieżności, ponieważ nie jest konieczne oczekiwanie na zakończenie całej iteracji. Stworzono w tym celu potokową wersję algorytmu zapisaną w funkcji *pipeGaussElimination()*. Przyjmuje ona jako argumenty rozmiar badanej macierzy oraz samą macierz. Wymagającym elementem jest zarządzanie wątkami programu. Utworzono więc pomocnicze tablice dynamiczne.

Tablica przechowująca postęp działania *doneTable* o wymiarach identycznych jak macierz obliczana jest na początku wypełniona samymi zerami. Każde kolejne działanie na polu macierzy zwiększa jego licznik o jeden. Po pierwszej (zerowej) iteracji cała tablica powinna być wypełniona jedynkami. Po kolejnej iteracji cała tablica z wyjątkiem pierwszego wiersza i kolumny wypełnia się dwójkami itd. Po zakończeniu programu tablica powinna wyglądać jak:

1	1	1	1	1	1
1	2	2	2	2	2
1	2	3	3	3	3
1	2	3	4	4	4
1	2	3	4	5	5

Tablica z kolejnością wykonywania *executeOrderTable* określa kolejne zadania – wiersze do obliczenia/wyeliminowania oraz iterację w której operacja ma być wykonana. Przykładowa zawartość:

Wiersz	0	1	2	3	4	1	2	3	4	2	3	4	3	4	4
Iteracja	0	0	0	0	0	1	1	1	1	2	2	2	3	3	4

```
void pipeGaussElimination(int n, float** AB)
{
    //tablica przechowująca postęp działania
    int** doneTable;
    doneTable = new int* [n];
    for (int j = 0; j < n; j++) doneTable[j] = new int[n + 1];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n + 1; j++) doneTable[i][j] = 0;
    }

    //tablica z kolejnością wykonywania
    int** executeOrderTable;
    executeOrderTable = new int* [(n * (1 + n)) / 2];
    for (int j = 0; j < (n * (1 + n)) / 2; j++) executeOrderTable[j] = new int[2];

    int tmp = 0;
    for (int k = 0; k < n; k++)
    {
        int var = k;
        while (var < n)
        {
            executeOrderTable[tmp][0] = var;    //który wiersz
            executeOrderTable[tmp][1] = k;      //która iteracja
            //cout << var << ": " << k << endl;;
            var++;
            tmp++;
        }
    }
    //tablice zostały przygotowane
```

Następnie zdefiniowana jest pętla realizująca przetwarzanie równoległe. Rozdziela ona zadania wg kolejności zapisanej w tabeli pomocniczej. Jeśli numer wiersza jest równy numerowi iteracji należy dokonać odpowiednich obliczeń. Na podstawie tego wiersza eliminowane będą wiersze kolejne. Nie można go jednak obliczyć dopóki nie zostaną dokonane na nim wszystkie poprzednie działania. Konieczne jest więc stworzenie pętli która będzie oczekiwała, aż wartość w tabeli *doneTable* będzie równa aktualnej iteracji. Okazało się, że jest to element newralgiczny, wpływający na działanie programu. W niektórych sytuacjach dochodzi do „zakleszczenia" programu, gdy wszystkie pozostałe wątki oczekują na wykonanie obliczeń przez pojedynczy wątek. Debugowanie programu pokazuje, że pomimo spełnienia warunku w poleceniu *if*, nie jest ustawiana flaga pozwalająca na przejście do dalszej części programu.

Rozwiązaniem okazało się być dodanie w poleceniu *else* wyrażenia *cout << ""*. Być może wywołanie podprocedur wpływa na „zresetowanie" wątku. Nie jest to jednak rozwiązanie które można zaakceptować jako realizujące założenia algorytmu.

Po etapie oczekiwania ustawiany jest stosunek *ratio* równy wartości pierwszego elementu rozpatrywanego w iteracji.

```
#pragma omp parallel for schedule(dynamic, 1) firstprivate(n) shared(AB, doneTable, executeOrderTable)
    for (int k = 0; k < (n*(1+n))/2; k++)
    {
        int row = executeOrderTable[k][0]; //wiersz przydzielony wątkowi
        int iteration = executeOrderTable[k][1]; //wiersz przez który wątek będzie dzielił - nr
iteracji



        //cout << row << iteration << endl;
        if (row == iteration) //jeśli wiersz jest pierwszy w kolejnej iteracji
        {
            //obliczanie
            //jesli element nie został jeszcze obliczony przez poprzednie iteracje to czekaj
            bool flag = true;
            do
            {
                //TU TKWI PROBLEM
                if (doneTable[row][row] == iteration)
                {
                    flag = false;
                }
                else
                {
                    //!!!!!!!!!!!!!!!!!cout << 'z' << doneTable[row][row] << endl; - z tą komendą działa
                    //cout << 'z'; //- z tą też
                    //iteration = iteration; //- z tą nie
                    //if (doneTable[row][row] == iteration)         //Z TYM
                    //{                                             //TEŻ
                    //    flag = false;                             //NIE
                    //}                                             //DZIAŁA
                    //int i;    // też nie
                    //int i = doneTable[row][row] / iteration;  //też nie
                    //cout; //też nie
                    cout << ""; // działa
                    flag = true;
                }
            } while (flag);
            float ratio = AB[row][row];
```

Po uzyskaniu wartości *ratio* następuje dzielenie wszystkich elementów w pierwszym wierszu przez tę liczbę. Tu również konieczne jest sprawdzanie, czy elementy zostały już obliczone przez poprzednie iteracje. Po obliczeniu pola jego status w tablicy *doneTable* jest zwiększany.

```
for (int i = row; i < n + 1; i++)
            {
                //if (i == row) cout << 'y' << doneTable[row][i] << endl;
                //jesli element nie został jeszcze obliczony przez poprzednie iteracje to czekaj
                bool flag;
                do
                {
                    flag = true;
                    if (doneTable[row][i] == iteration)
                    {
                        flag = false;
                        //cout << 'y';
                    }
                    else
                    {
                        flag = true;
                        //cout << 'x';
                        //cout << iteration << endl;
                    }
                } while (flag);
                AB[row][i] = AB[row][i] / ratio;
                doneTable[row][i]=iteration+1;    //obliczony element otrzymuje status o jeden większy
niż poprzednio
                //if (i==row) cout << 'x' << doneTable[row][i] << endl;
                //#pragma omp critical
                //{
                //    printMatrix(n, doneTable, row, i);
                //};
            }
        }
```

Dla pozostałych wierszy w iteracji zachodzi eliminacja. Tu również znajdują się dwie pętle oczekujące, które wpływają na stabilność funkcji. Brak polecenia *cout << ""* stwarza ryzyko zawieszenia się programu. Obliczany jest stosunek na podstawie ilorazu pierwszego elementu w eliminowanym wierszu oraz pierwszego elementu w iteracji. Od pozostałych elementów odejmowany jest iloczyn obliczonego stosunku i odpowiedniego elementu z pierwszego wiersza iteracji. Po obliczeniu status elementu jest modyfikowany w tabeli *doneTable*.

```
        else
        {
            //eliminacja
            //dopóki elementy nie zostały obliczone przez poprzednie iteracje to czekaj
            bool flag = true;
            do
            {
                if (doneTable[row][iteration] == iteration && doneTable[iteration][iteration] ==
iteration+1)
                {
                    flag = false;
                    //cout << 'y';
                }
                else
                {
                    //TU UTYKAJĄ WĄTKI
                    flag = true;
                    //cout << 'x';
                    //cout << iteration+1 << doneTable[iteration][iteration] << endl;   //widoczna
różnica, przyczyna oczekiwania   //TEŻ WYMAGANE DO DZIAŁANIA
                    cout << "";
                }
            } while (flag);
            float ratio = AB[row][iteration] / AB[iteration][iteration];
            //dla wszystkich pozostałych elementów w wierszu
            for (int j = iteration; j < n + 1; j++)
            {
                //dopóki elementy nie zostały obliczone przez poprzednie iteracje to czekaj
                bool flag;
                do
                {
                    if (doneTable[row][j] == iteration && doneTable[iteration][j] == iteration + 1)
                    {
                        flag = false;
                        //cout << 'y';
                    }
                    else
                    {
                        flag = true;
                        //cout << 'x';
                        //cout << iteration << endl;
                    }
                } while (flag);
                AB[row][j] = AB[row][j] - ratio * AB[iteration][j];
                doneTable[row][j]=iteration+1;
                //#pragma omp critical
                //{
                //    printMatrix(n, doneTable, row, j);
                //}
            }
        }

    }
```

Po zakończeniu działania funkcja usuwa dynamicznie utworzone tablice pomocnicze.

```
    for (int j = 0; j < n; j++) delete[] doneTable[j];
    delete[] doneTable;
    for (int j = 0; j < (n * (1 + n)) / 2; j++) delete[] executeOrderTable[j];
    delete[] executeOrderTable;
}
```

Otrzymane wyniki

W związku z przedłużającym się badaniem oraz niesatysfakcjonującymi wynikami przerwano działanie procedury przed jej całkowitym ukończeniem.

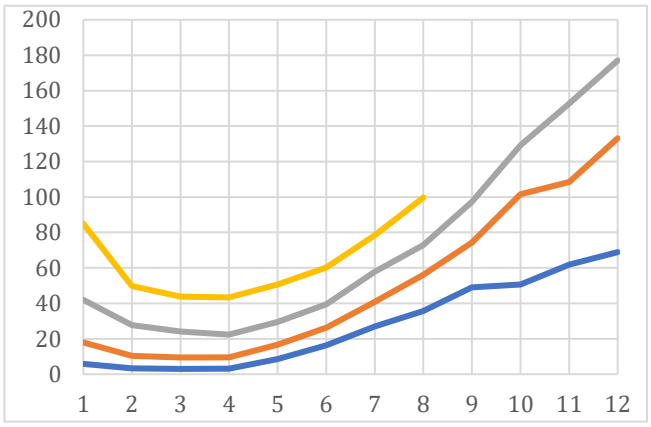
Średni czas wykonania
Oś pozioma: liczba wątków 1-12
Oś pionowa: średni czas w s

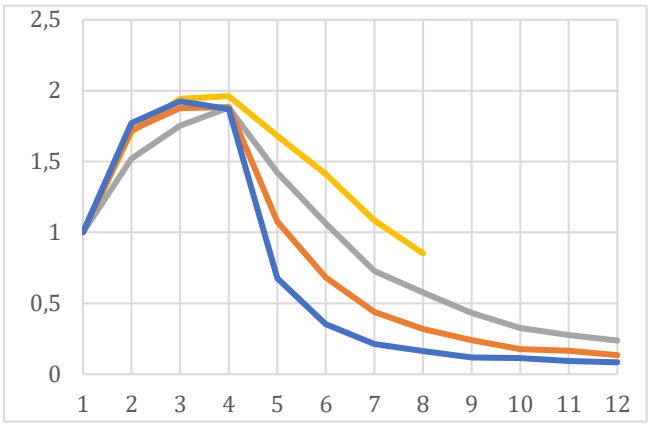
Rozmiar macierzy
4000
3200
2400
1600

Przyspieszenie
Oś pozioma: liczba wątków 1-12
Oś pionowa: przyspieszenie

100% wyników

Wątków	1	2	3	4	5	6	7	8	9	10	11	12
Średni czas	85,03	49,79	43,76	43,35	50,59	60,18	78,37	99,72				
Przyspieszenie	1,00	1,71	1,94	1,96	1,68	1,41	1,08	0,85				
Średni czas	42,06	27,66	24,01	22,37	29,60	39,48	57,86	72,82	97,16	129,15	152,61	177,06
Przyspieszenie	1,00	1,52	1,75	1,88	1,42	1,07	0,73	0,58	0,43	0,33	0,28	0,24
Średni czas	17,93	10,42	9,55	9,52	16,66	26,27	40,92	56,14	74,33	101,55	108,48	133,14
Przyspieszenie	1,00	1,72	1,88	1,88	1,08	0,68	0,44	0,32	0,24	0,18	0,17	0,13
Średni czas	5,78	3,26	3,00	3,09	8,55	16,33	27,07	35,56	48,94	50,67	61,84	68,89
Przyspieszenie	1,00	1,77	1,92	1,87	0,68	0,35	0,21	0,16	0,12	0,11	0,09	0,08





Działanie algorytmy asynchronicznego pozwala na uzyskanie odrobinę lepszego przyspieszenia. Jednak wykorzystanie liczby wątków większej niż wynika to z budowy procesora skutkuje znacznym spadkiem wydajności i wydłużeniem czasu działania algorytmu w porównaniu z wersją jednowątkową. Większa liczba danych skutkuje że spadek ten jest mniej gwałtowny.

Uporządkowane rosnąco czasy obliczania synchronicznego [ms] dla macierzy rozmiaru 1600

Lp.	Wątków											
	1	2	3	4	5	6	7	8	9	10	11	12
1	1251	795	742	728	747	749	740	735	737	758	755	740
2	1252	812	743	737	748	761	746	735	737	759	756	742
3	1253	813	747	742	750	763	748	744	738	760	757	748
4	1254	813	747	744	750	764	748	746	745	760	758	750
5	1254	816	748	746	758	765	749	748	748	760	759	752
6	1255	823	749	747	758	767	750	749	751	763	759	753
7	1256	826	750	747	759	770	753	749	752	763	760	754
8	1257	827	751	749	760	771	754	750	752	765	760	754
9	1258	830	751	749	761	772	754	750	753	765	762	754
10	1258	830	752	749	761	776	757	751	755	766	764	754
11	1258	832	753	749	764	777	757	752	755	766	765	755
12	1258	833	753	750	765	777	758	753	756	767	765	755
13	1259	835	754	751	765	778	759	754	756	767	765	756
14	1259	836	755	751	768	782	760	755	756	767	767	756
15	1259	838	755	751	768	783	761	756	757	769	768	756
16	1261	839	755	751	768	783	761	756	758	769	768	759
17	1262	839	755	752	769	783	761	757	758	769	768	759
18	1262	841	756	752	769	784	761	757	759	769	769	760
19	1263	842	757	752	769	784	762	757	759	770	769	760
20	1264	844	757	752	771	786	763	757	759	771	770	761
21	1264	844	757	753	773	786	763	757	760	771	770	761
22	1264	844	757	753	773	789	765	759	761	771	770	761
23	1265	844	758	755	775	790	766	760	761	772	771	761
24	1266	845	758	755	775	790	766	761	761	772	771	762
25	1266	845	760	756	776	790	767	762	762	772	772	762
26	1266	845	760	757	777	791	768	762	762	772	774	763
27	1266	849	760	757	778	791	769	763	762	774	775	764
28	1266	849	760	757	778	792	769	764	765	774	775	765
29	1266	850	761	758	778	792	770	764	766	774	775	765
30	1266	851	761	758	781	792	770	765	766	775	775	766
31	1267	852	762	759	781	794	770	765	767	775	776	766
32	1267	853	762	759	781	794	770	765	767	775	778	766
33	1267	853	762	760	786	794	771	766	767	776	778	767
34	1268	854	762	760	787	795	772	766	767	776	778	767
35	1269	854	763	760	788	795	773	767	768	776	778	767
36	1269	855	763	760	788	796	774	767	768	777	778	767
37	1269	856	763	761	792	796	774	768	768	778	778	767
38	1270	857	763	761	793	797	774	768	768	778	780	767
39	1270	858	764	761	800	799	774	768	770	778	780	767
40	1270	858	764	761	802	799	775	768	770	778	780	768
41	1271	858	764	762	805	800	776	768	770	780	781	768
42	1271	859	764	762	809	800	776	768	770	780	782	768
43	1272	859	764	762	811	801	777	769	770	780	782	768
44	1272	866	765	763	811	801	777	770	771	781	782	768
45	1273	867	765	764	812	801	777	770	771	781	783	768
46	1273	867	765	764	813	801	777	770	771	781	783	769
47	1273	869	765	764	816	802	777	772	771	781	784	769
48	1273	869	765	765	818	802	778	772	771	782	784	769
49	1274	870	766	765	819	802	778	773	772	782	784	769
50	1274	870	766	765	819	803	779	774	772	782	785	769
51	1274	873	766	765	819	803	779	774	772	783	786	770
52	1274	873	766	765	819	804	780	774	773	783	786	770
53	1275	874	766	766	820	804	781	774	773	783	788	770
54	1275	874	767	766	820	804	781	774	773	783	789	771
55	1276	875	767	766	822	805	781	775	774	785	789	771
56	1276	875	768	766	822	805	782	775	774	786	789	771
57	1276	876	768	766	823	806	782	776	774	786	790	771
58	1277	877	769	766	824	806	783	777	775	786	791	772
59	1279	877	769	767	825	806	783	777	776	787	792	772
60	1280	878	769	767	825	806	784	778	777	787	793	772
61	1280	880	769	768	825	806	784	779	778	788	795	772
62	1281	880	769	768	827	807	784	779	778	788	795	772
63	1283	880	770	769	827	807	784	780	779	788	797	773
64	1283	881	770	770	829	807	784	780	779	789	797	773
65	1283	881	771	771	829	808	784	782	780	789	798	774
66	1284	882	771	772	829	808	785	784	781	790	798	775
67	1287	884	771	773	830	809	786	784	781	790	798	775
68	1289	885	772	773	831	810	786	785	781	791	799	775
69	1289	886	772	773	832	811	786	785	781	792	800	776
70	1294	889	773	773	833	811	786	786	782	792	803	776
71	1294	890	775	774	834	811	787	786	782	793	803	776
72	1296	891	781	774	834	812	787	787	783	793	806	777
73	1299	891	781	774	834	812	787	788	785	794	806	777
74	1301	892	781	776	834	813	788	788	785	797	806	777
75	1302	896	782	777	834	814	788	789	786	798	808	778
76	1306	897	783	777	835	815	789	789	786	799	809	780
77	1309	899	784	779	835	816	790	790	787	800	817	781
78	1311	899	785	780	838	817	791	792	789	801	818	783
79	1316	899	789	781	840	818	791	797	794	806	821	783
80	1319	900	791	784	842	819	792	797	794	806	821	785
81	1321	901	793	785	852	821	793	797	795	807	822	787
82	1324	904	797	790	853	821	793	798	803	813	828	787
83	1325	908	799	794	858	821	794	804	804	817	832	791
84	1326	910	803	795	861	822	795	809	805	820	844	796
85	1336	916	808	795	861	822	804	810	807	830	851	796
86	1358	920	823	797	863	822	805	811	811	831	856	799
87	1371	923	827	799	870	823	806	813	813	835	858	800
88	1376	923	827	799	873	823	808	815	814	837	859	801
89	1379	927	827	801	876	828	809	817	819	851	867	802
90	1380	928	840	814	882	831	811	817	819	860	880	807
91	1388	933	841	818	891	833	820	818	820	860	884	807
92	1393	934	844	819	892	845	838	831	825	862	889	815
93	1402	935	845	831	893	852	844	835	834	878	897	815
94	1435	940	860	840	895	859	845	839	839	887	898	822
95	1454	948	876	840	898	861	849	839	842	902	926	835
96	1471	966	909	843	903	866	853	861	844	918	933	839
97	1552	978	928	853	919	866	861	867	861	950	996	843
98	1591	998	938	929	1002	869	864	868	881	1044	1058	847
99	2188	1082	940	943	1136	879	871	869	882	1169	1102	853
100	4545	1352	1054	971	1888	905	1047	937	936	4438	1177	1036

Uporządkowane rosnąco czasy obliczania synchronicznego [ms] dla macierzy rozmiaru 2400

Lp.	Wątków											
	1	2	3	4	5	6	7	8	9	10	11	12
1	4188	2872	2514	2509	2511	2530	2519	2514	2527	2544	2541	2541
2	4199	2881	2524	2510	2513	2532	2533	2533	2537	2553	2544	2549
3	4202	2899	2528	2516	2515	2536	2538	2537	2546	2556	2544	2556
4	4204	2903	2535	2519	2520	2537	2542	2542	2550	2557	2546	2556
5	4207	2906	2536	2520	2520	2541	2543	2543	2550	2559	2552	2558
6	4210	2908	2536	2523	2523	2542	2544	2545	2550	2559	2555	2559
7	4211	2911	2537	2525	2525	2542	2546	2545	2552	2561	2558	2559
8	4211	2912	2537	2527	2532	2543	2546	2550	2558	2562	2558	2560
9	4213	2913	2538	2528	2534	2544	2547	2552	2559	2564	2560	2562
10	4214	2915	2539	2530	2535	2544	2547	2552	2562	2564	2561	2563
11	4215	2916	2539	2530	2536	2545	2548	2553	2563	2566	2561	2564
12	4216	2919	2540	2532	2537	2546	2550	2554	2564	2566	2562	2565
13	4217	2921	2540	2537	2543	2546	2551	2554	2564	2568	2562	2566
14	4218	2923	2541	2539	2545	2549	2551	2554	2565	2569	2562	2567
15	4218	2923	2541	2540	2547	2550	2551	2555	2566	2570	2563	2571
16	4218	2929	2541	2541	2547	2550	2552	2557	2566	2570	2563	2572
17	4219	2930	2543	2541	2548	2551	2552	2558	2567	2571	2564	2574
18	4220	2930	2544	2543	2548	2553	2554	2558	2567	2572	2567	2574
19	4223	2931	2544	2544	2551	2553	2554	2559	2568	2572	2567	2575
20	4224	2933	2547	2546	2551	2553	2554	2559	2571	2573	2567	2575
21	4224	2936	2547	2547	2552	2554	2555	2560	2573	2573	2568	2575
22	4227	2937	2547	2547	2553	2554	2557	2561	2574	2574	2569	2575
23	4227	2939	2549	2549	2554	2555	2558	2561	2574	2574	2569	2575
24	4227	2939	2550	2550	2556	2555	2559	2566	2575	2575	2571	2575
25	4228	2939	2550	2550	2557	2558	2561	2566	2575	2576	2571	2575
26	4228	2941	2550	2551	2557	2558	2561	2566	2575	2576	2573	2576
27	4229	2942	2551	2551	2558	2559	2561	2568	2576	2576	2574	2576
28	4230	2942	2552	2552	2560	2559	2562	2569	2576	2578	2575	2577
29	4233	2943	2552	2554	2561	2559	2562	2569	2577	2578	2576	2577
30	4234	2943	2552	2555	2561	2561	2563	2570	2577	2580	2576	2577
31	4238	2944	2553	2555	2562	2561	2563	2571	2577	2581	2578	2577
32	4238	2946	2554	2556	2563	2561	2564	2571	2578	2582	2579	2579
33	4240	2947	2554	2556	2564	2561	2564	2572	2578	2583	2580	2579
34	4241	2948	2555	2558	2564	2562	2564	2573	2579	2583	2580	2579
35	4242	2949	2555	2559	2564	2562	2565	2573	2579	2583	2581	2580
36	4247	2954	2556	2560	2565	2563	2568	2575	2579	2585	2583	2581
37	4247	2954	2556	2560	2565	2564	2569	2575	2580	2586	2583	2581
38	4250	2956	2556	2560	2565	2564	2569	2575	2580	2586	2584	2582
39	4255	2961	2557	2561	2566	2566	2570	2577	2582	2587	2585	2583
40	4255	2963	2557	2561	2568	2566	2570	2577	2583	2587	2585	2585
41	4259	2964	2557	2562	2569	2566	2570	2577	2584	2587	2586	2585
42	4262	2966	2559	2562	2569	2567	2571	2578	2585	2589	2586	2585
43	4263	2966	2559	2563	2571	2567	2571	2580	2585	2589	2586	2586
44	4279	2969	2561	2565	2573	2568	2571	2580	2587	2590	2587	2586
45	4280	2972	2562	2565	2576	2568	2572	2580	2587	2590	2587	2588
46	4284	2973	2562	2567	2577	2568	2572	2581	2587	2591	2588	2590
47	4285	2975	2563	2570	2579	2569	2574	2581	2588	2591	2588	2590
48	4287	2976	2564	2571	2579	2571	2574	2582	2588	2592	2588	2590
49	4294	2977	2564	2572	2580	2572	2575	2584	2588	2592	2589	2591
50	4315	2981	2565	2574	2580	2572	2575	2584	2591	2593	2589	2591
51	4322	2982	2565	2575	2580	2574	2575	2585	2592	2593	2589	2591
52	4361	2984	2565	2576	2580	2575	2576	2587	2593	2593	2589	2593
53	4403	2986	2566	2578	2581	2579	2579	2587	2593	2595	2590	2595
54	4407	2987	2567	2578	2582	2579	2579	2588	2593	2599	2591	2596
55	4408	2989	2567	2580	2582	2581	2580	2589	2595	2600	2592	2596
56	4410	2989	2570	2580	2583	2582	2580	2590	2595	2600	2592	2596
57	4420	2991	2570	2581	2584	2582	2581	2592	2596	2602	2593	2596
58	4470	2992	2572	2582	2584	2582	2582	2592	2597	2603	2597	2597
59	4481	2994	2573	2584	2590	2585	2583	2593	2597	2604	2597	2598
60	4484	2994	2577	2587	2590	2585	2583	2593	2598	2606	2598	2600
61	4492	2994	2579	2589	2592	2588	2585	2595	2600	2607	2599	2600
62	4492	2995	2583	2590	2593	2588	2587	2599	2603	2610	2600	2600
63	4503	2997	2584	2590	2593	2589	2587	2601	2604	2611	2602	2602
64	4506	2998	2586	2592	2593	2591	2590	2602	2607	2612	2604	2602
65	4507	3003	2588	2594	2601	2595	2592	2602	2609	2614	2606	2606
66	4509	3005	2589	2594	2608	2597	2593	2603	2609	2614	2608	2606
67	4510	3008	2590	2595	2613	2599	2594	2605	2611	2616	2609	2609
68	4522	3009	2590	2598	2619	2599	2595	2611	2612	2616	2610	2609
69	4530	3010	2592	2603	2621	2601	2596	2614	2612	2618	2610	2610
70	4539	3015	2597	2610	2623	2602	2597	2615	2613	2618	2618	2614
71	4567	3017	2597	2611	2639	2603	2597	2616	2614	2618	2622	2614
72	4578	3020	2599	2611	2645	2605	2599	2616	2617	2620	2623	2616
73	4579	3030	2603	2613	2652	2607	2600	2618	2618	2620	2624	2623
74	4586	3033	2605	2620	2653	2607	2601	2620	2621	2620	2624	2627
75	4586	3033	2607	2623	2658	2610	2606	2621	2621	2622	2626	2632
76	4594	3034	2609	2625	2659	2610	2606	2633	2622	2622	2626	2634
77	4596	3035	2610	2633	2666	2615	2606	2640	2622	2623	2629	2635
78	4603	3036	2611	2633	2669	2615	2606	2649	2624	2625	2634	2635
79	4603	3038	2611	2635	2678	2620	2607	2649	2626	2627	2637	2640
80	4605	3058	2612	2636	2685	2621	2612	2650	2633	2627	2645	2640
81	4611	3060	2616	2639	2691	2621	2617	2660	2635	2628	2648	2647
82	4624	3078	2617	2646	2694	2626	2628	2691	2637	2629	2649	2648
83	4634	3088	2619	2646	2697	2627	2628	2704	2639	2629	2653	2659
84	4639	3093	2624	2650	2700	2628	2628	2710	2640	2631	2663	2660
85	4668	3097	2625	2654	2701	2635	2628	2714	2640	2635	2672	2662
86	4672	3099	2629	2675	2703	2636	2639	2721	2659	2635	2681	2664
87	4722	3121	2633	2675	2711	2642	2642	2730	2659	2645	2685	2668
88	4750	3122	2635	2689	2715	2648	2647	2734	2660	2646	2687	2672
89	4829	3134	2646	2690	2720	2651	2648	2780	2667	2647	2699	2685
90	4901	3146	2663	2693	2737	2652	2648	2782	2679	2648	2707	2697
91	4951	3148	2728	2721	2754	2673	2652	2790	2726	2658	2714	2699
92	4957	3151	2729	2722	2766	2698	2658	2799	2751	2666	2732	2707
93	5153	3156	2731	2729	2769	2704	2667	2800	2752	2686	2734	2720
94	5187	3188	2733	2737	2784	2719	2682	2843	2777	2686	2748	2762
95	5429	3195	2735	2760	2800	2721	2716	2873	2780	2738	2749	2777
96	6022	3207	2745	2821	2816	2725	2723	2884	2803	2782	2759	2783
97	6344	3212	2748	2849	2818	2732	2735	2910	2815	2799	2781	2828
98	9200	3217	2781	2877	2854	2738	2750	2935	2845	2831	2807	2841
99	11563	3289	3003	2939	2898	2761	2805	2941	2965	2904	2889	2966
100	13496	3338	3252	3142	2990	2922	2817	3042	3067	2913	3031	2989

Uporządkowane rosnąco czasy obliczania synchronicznego [ms] dla macierzy rozmiaru 3200

Lp.	Wątków											
	1	2	3	4	5	6	7	8	9	10	11	12
1	9890	6658	5843	5849	5845	5885	5866	5883	5887	5873	5887	5952
2	9891	6658	5845	5861	5848	5886	5876	5892	5889	5888	5890	5986
3	9892	6661	5845	5863	5863	5888	5881	5894	5893	5892	5899	6003
4	9898	6661	5855	5863	5869	5892	5882	5894	5900	5895	5901	6012
5	9909	6680	5856	5866	5872	5898	5883	5903	5906	5897	5901	6109
6	9911	6685	5857	5877	5872	5901	5883	5903	5914	5898	5905	6130
7	9913	6686	5863	5877	5873	5902	5887	5905	5922	5905	5905	6146
8	9914	6686	5866	5878	5875	5904	5888	5905	5922	5918	5905	6150
9	9916	6693	5869	5881	5876	5904	5888	5907	5923	5918	5908	6150
10	9918	6694	5870	5882	5877	5907	5888	5907	5928	5919	5916	6152
11	9923	6694	5873	5885	5880	5908	5890	5907	5928	5920	5918	6157
12	9929	6695	5873	5886	5880	5910	5890	5907	5937	5920	5920	6163
13	9937	6705	5874	5887	5881	5911	5897	5910	5938	5921	5922	6169
14	9937	6713	5874	5890	5881	5912	5897	5911	5938	5921	5924	6172
15	9939	6713	5875	5890	5883	5915	5898	5914	5938	5922	5924	6173
16	9941	6717	5876	5892	5886	5916	5898	5914	5939	5924	5925	6173
17	9942	6726	5880	5894	5886	5917	5900	5917	5939	5924	5926	6178
18	9942	6730	5884	5896	5891	5918	5901	5917	5939	5926	5926	6179
19	9947	6734	5886	5896	5891	5919	5903	5920	5940	5928	5926	6183
20	9947	6745	5888	5897	5895	5919	5903	5922	5940	5929	5928	6187
21	9947	6748	5888	5899	5897	5924	5905	5922	5946	5929	5928	6187
22	9949	6759	5888	5901	5897	5924	5905	5926	5947	5930	5930	6190
23	9951	6760	5897	5901	5898	5926	5906	5928	5949	5931	5933	6190
24	9952	6760	5898	5902	5901	5926	5906	5929	5951	5933	5935	6190
25	9954	6765	5899	5902	5901	5927	5908	5929	5951	5933	5938	6195
26	9955	6766	5904	5904	5901	5928	5909	5929	5951	5935	5939	6196
27	9956	6774	5907	5904	5902	5932	5910	5931	5952	5935	5943	6197
28	9956	6774	5909	5905	5903	5937	5912	5933	5952	5936	5946	6203
29	9957	6785	5914	5905	5906	5939	5915	5934	5952	5937	5946	6203
30	9957	6795	5917	5906	5911	5939	5915	5934	5953	5940	5947	6205
31	9961	6796	5917	5906	5911	5942	5915	5934	5953	5940	5947	6205
32	9962	6797	5918	5906	5915	5945	5916	5937	5954	5940	5950	6206
33	9962	6799	5919	5907	5915	5945	5917	5938	5955	5941	5952	6208
34	9962	6801	5919	5907	5916	5945	5917	5938	5956	5942	5953	6208
35	9962	6803	5921	5908	5916	5945	5918	5939	5958	5942	5955	6212
36	9962	6812	5924	5911	5920	5947	5919	5940	5962	5942	5955	6213
37	9964	6817	5924	5912	5921	5949	5922	5941	5963	5942	5957	6213
38	9966	6817	5927	5917	5921	5950	5923	5941	5963	5943	5958	6214
39	9973	6823	5931	5918	5924	5953	5923	5941	5965	5944	5962	6215
40	9974	6827	5931	5921	5925	5954	5925	5944	5967	5945	5962	6215
41	9975	6836	5933	5922	5925	5955	5926	5945	5969	5946	5962	6218
42	9979	6846	5934	5924	5928	5956	5929	5946	5969	5946	5963	6219
43	9981	6847	5934	5932	5929	5957	5932	5946	5969	5949	5963	6224
44	9981	6847	5937	5932	5930	5958	5933	5947	5971	5951	5968	6225
45	9982	6849	5938	5938	5930	5961	5936	5948	5971	5952	5969	6228
46	9983	6852	5938	5941	5934	5962	5941	5951	5972	5954	5969	6229
47	9984	6853	5939	5941	5935	5965	5946	5951	5973	5957	5970	6233
48	9987	6854	5940	5943	5935	5966	5948	5954	5974	5960	5972	6233
49	9987	6856	5941	5946	5936	5967	5951	5955	5974	5962	5972	6234
50	9989	6859	5943	5948	5943	5970	5953	5957	5979	5963	5973	6235
51	9994	6868	5947	5949	5944	5972	5953	5958	5985	5964	5973	6243
52	9996	6873	5947	5950	5947	5972	5956	5959	5988	5966	5975	6244
53	10000	6875	5950	5951	5949	5977	5956	5960	5989	5967	5978	6245
54	10002	6879	5952	5955	5949	5977	5957	5964	5991	5967	5979	6247
55	10007	6883	5958	5959	5955	5977	5958	5965	5996	5968	5980	6247
56	10007	6883	5959	5961	5956	5981	5959	5968	5999	5970	5981	6248
57	10009	6884	5964	5962	5961	5983	5960	5968	6006	5970	5981	6248
58	10013	6888	5964	5962	5962	5990	5961	5971	6006	5973	5983	6253
59	10017	6892	5966	5964	5963	5990	5961	5972	6007	5979	5984	6261
60	10017	6900	5967	5965	5963	5992	5965	5972	6008	5980	5986	6265
61	10018	6903	5974	5967	5965	5996	5968	5973	6008	5984	5988	6265
62	10030	6925	5975	5967	5967	6001	5968	5974	6008	5989	5991	6269
63	10032	6929	5976	5969	5968	6001	5971	5975	6012	5989	5991	6270
64	10035	6946	5977	5971	5973	6003	5972	5976	6018	5996	5993	6270
65	10040	6946	5981	5974	5973	6004	5976	5980	6018	5998	5995	6272
66	10040	6951	5983	5977	5974	6005	5979	5981	6019	5999	6003	6279
67	10042	6954	5989	5979	5976	6007	5981	5987	6019	6005	6009	6288
68	10048	6955	5990	5983	5978	6008	5981	5987	6020	6006	6013	6293
69	10051	6955	5991	5987	5979	6018	5983	5987	6023	6006	6015	6303
70	10051	6978	6006	5996	5981	6019	5986	5987	6029	6006	6018	6303
71	10057	6979	6016	6005	5996	6024	5987	5988	6029	6008	6021	6305
72	10063	6984	6017	6007	5999	6024	5988	5991	6030	6025	6021	6309
73	10067	6988	6027	6010	6005	6027	5994	5992	6031	6027	6027	6328
74	10068	6990	6031	6018	6009	6027	5999	5996	6032	6030	6032	6338
75	10088	7003	6036	6029	6035	6029	6005	6003	6052	6038	6032	6374
76	10120	7006	6041	6030	6036	6035	6006	6014	6052	6038	6039	6387
77	10121	7009	6041	6031	6039	6037	6020	6016	6056	6044	6041	6405
78	10123	7016	6049	6041	6040	6040	6020	6018	6068	6048	6050	6407
79	10129	7025	6055	6061	6044	6041	6023	6037	6070	6049	6052	6421
80	10229	7030	6063	6062	6045	6045	6051	6038	6073	6050	6053	6426
81	10265	7031	6064	6065	6049	6045	6055	6039	6076	6061	6056	6433
82	10282	7037	6067	6073	6051	6052	6067	6061	6131	6085	6062	6435
83	10349	7065	6083	6074	6051	6087	6069	6065	6142	6094	6068	6449
84	10384	7067	6100	6077	6053	6091	6071	6080	6145	6106	6080	6459
85	10390	7079	6108	6094	6086	6099	6074	6081	6155	6120	6084	6467
86	10412	7085	6126	6109	6099	6133	6083	6102	6157	6128	6089	6473
87	10423	7107	6147	6127	6109	6133	6116	6119	6162	6159	6102	6475
88	10424	7108	6154	6151	6139	6140	6170	6128	6163	6180	6118	6517
89	10446	7119	6184	6157	6144	6157	6192	6133	6178	6197	6122	6531
90	10465	7131	6201	6158	6153	6160	6193	6150	6209	6244	6144	6548
91	10499	7188	6213	6168	6170	6176	6205	6168	6212	6248	6159	6559
92	10528	7192	6234	6171	6179	6200	6211	6176	6222	6259	6232	6582
93	10549	7200	6247	6178	6194	6216	6212	6181	6226	6266	6236	6589
94	10576	7216	6259	6180	6194	6236	6212	6226	6229	6270	6266	6658
95	10577	7223	6273	6213	6236	6247	6228	6253	6243	6283	6300	6826
96	10578	7235	6281	6269	6271	6303	6250	6255	6248	6299	6301	6841
97	10583	7284	6366	6285	6290	6318	6277	6295	6252	6303	6314	6873
98	10712	7314	6384	6295	6304	6347	6362	6381	6299	6350	6318	7262
99	10721	7349	6565	6306	6307	6368	6382	6412	6396	6360	6400	7305
100	10867	7586	7140	6379	6359	6384	6643	6602	6437	6416	6487	7800

Uporządkowane rosnąco czasy obliczania synchronicznego [ms] dla macierzy rozmiaru 4000

Lp.	Wątków											
	1	2	3	4	5	6	7	8	9	10	11	12
1	21088	13666	11386	11285	11345	11314	11332	11311	11357	11356	11381	11381
2	21095	13675	11413	11291	11352	11335	11341	11341	11374	11374	11383	11391
3	21099	13713	11439	11304	11364	11336	11347	11355	11387	11382	11398	11392
4	21101	13728	11446	11308	11370	11352	11362	11359	11389	11384	11407	11395
5	21107	13728	11458	11310	11371	11363	11362	11362	11390	11394	11408	11398
6	21110	13750	11468	11313	11383	11371	11367	11374	11390	11395	11411	11406
7	21116	13758	11476	11313	11385	11373	11367	11391	11394	11396	11415	11411
8	21116	13765	11482	11328	11391	11373	11368	11392	11397	11415	11417	11424
9	21119	13775	11483	11338	11394	11377	11370	11397	11400	11417	11419	11425
10	21128	13782	11496	11340	11401	11377	11380	11400	11404	11418	11423	11426
11	21137	13783	11527	11342	11408	11385	11385	11405	11411	11426	11424	11431
12	21137	13812	11529	11349	11423	11385	11391	11406	11411	11426	11429	11431
13	21137	13828	11533	11351	11428	11396	11392	11408	11412	11426	11433	11432
14	21140	13833	11548	11354	11431	11397	11394	11416	11412	11427	11434	11440
15	21142	13838	11549	11367	11436	11398	11396	11419	11413	11428	11435	11448
16	21147	13843	11551	11370	11441	11405	11403	11422	11421	11428	11435	11449
17	21151	13846	11561	11380	11444	11406	11403	11423	11422	11430	11439	11451
18	21155	13850	11562	11388	11447	11407	11406	11427	11424	11432	11441	11458
19	21164	13858	11563	11388	11454	11407	11407	11429	11437	11433	11447	11460
20	21171	13861	11563	11389	11456	11407	11408	11438	11438	11433	11450	11460
21	21175	13861	11563	11399	11456	11409	11410	11438	11438	11437	11452	11464
22	21177	13868	11563	11403	11457	11412	11412	11439	11445	11438	11452	11464
23	21178	13879	11568	11404	11459	11413	11414	11440	11446	11440	11453	11465
24	21183	13880	11569	11410	11466	11415	11414	11443	11446	11440	11455	11466
25	21189	13883	11572	11414	11467	11423	11415	11443	11453	11441	11460	11476
26	21189	13889	11577	11421	11468	11425	11415	11444	11459	11441	11460	11480
27	21194	13894	11578	11423	11476	11426	11418	11446	11463	11443	11463	11483
28	21194	13904	11581	11424	11483	11429	11424	11447	11465	11444	11464	11487
29	21209	13909	11583	11441	11484	11431	11425	11447	11466	11450	11465	11489
30	21209	13910	11588	11444	11504	11432	11430	11449	11468	11452	11469	11494
31	21215	13928	11594	11446	11504	11433	11430	11451	11468	11453	11475	11497
32	21221	13943	11595	11447	11509	11433	11430	11451	11470	11457	11480	11498
33	21229	13947	11600	11448	11509	11434	11432	11452	11474	11458	11480	11499
34	21240	13949	11601	11448	11510	11437	11433	11465	11479	11458	11480	11499
35	21247	13949	11602	11449	11510	11439	11434	11466	11480	11460	11486	11505
36	21247	13959	11602	11453	11520	11441	11442	11466	11480	11464	11491	11507
37	21249	13962	11604	11468	11521	11441	11444	11472	11488	11466	11491	11507
38	21250	13964	11613	11468	11524	11443	11450	11476	11489	11469	11492	11510
39	21250	13974	11613	11476	11540	11448	11457	11477	11492	11472	11493	11525
40	21257	13981	11621	11479	11541	11452	11458	11479	11492	11477	11493	11534
41	21293	13989	11622	11485	11542	11455	11466	11483	11497	11485	11500	11537
42	21296	13994	11622	11490	11551	11455	11468	11486	11497	11487	11505	11540
43	21302	14000	11626	11493	11552	11456	11470	11487	11498	11487	11507	11543
44	21313	14004	11626	11497	11553	11457	11475	11492	11502	11489	11511	11543
45	21317	14005	11633	11503	11570	11458	11475	11497	11504	11491	11513	11547
46	21319	14009	11636	11504	11574	11459	11477	11498	11505	11493	11515	11549
47	21323	14011	11636	11510	11580	11465	11488	11498	11512	11513	11516	11557
48	21329	14013	11641	11531	11587	11465	11491	11498	11512	11516	11519	11560
49	21343	14023	11644	11545	11588	11468	11494	11505	11525	11518	11524	11564
50	21374	14024	11644	11552	11592	11470	11495	11506	11535	11518	11526	11565
51	21389	14034	11650	11580	11593	11473	11496	11513	11539	11531	11529	11568
52	21403	14034	11654	11582	11593	11488	11498	11523	11540	11531	11534	11574
53	21405	14044	11655	11582	11600	11500	11500	11526	11542	11531	11545	11575
54	21432	14045	11657	11583	11615	11501	11507	11530	11547	11535	11545	11576
55	21433	14046	11659	11601	11630	11506	11510	11535	11547	11539	11546	11580
56	21437	14049	11660	11612	11630	11510	11512	11537	11548	11541	11551	11580
57	21441	14055	11662	11615	11632	11514	11517	11555	11551	11542	11553	11581
58	21461	14055	11662	11618	11641	11519	11521	11570	11551	11548	11558	11587
59	21461	14061	11665	11623	11643	11527	11522	11575	11555	11551	11559	11588
60	21468	14061	11668	11623	11643	11528	11526	11576	11555	11551	11559	11593
61	21472	14062	11679	11626	11652	11531	11530	11580	11562	11556	11559	11606
62	21489	14064	11685	11626	11664	11533	11531	11589	11565	11563	11579	11618
63	21516	14068	11695	11631	11666	11534	11535	11591	11573	11571	11628	11625
64	21548	14090	11703	11634	11676	11537	11563	11609	11583	11582	11632	11628
65	21564	14099	11706	11635	11681	11540	11563	11617	11583	11588	11649	11630
66	21570	14104	11727	11646	11694	11561	11570	11623	11588	11589	11653	11635
67	21572	14106	11736	11647	11721	11573	11581	11623	11590	11611	11653	11637
68	21577	14108	11745	11656	11730	11575	11587	11624	11593	11611	11655	11639
69	21577	14125	11763	11671	11745	11590	11600	11624	11596	11612	11668	11644
70	21589	14125	11769	11675	11753	11592	11622	11633	11604	11614	11669	11647
71	21599	14130	11780	11676	11768	11594	11623	11639	11609	11620	11682	11650
72	21613	14139	11790	11684	11771	11597	11631	11642	11613	11654	11699	11664
73	21627	14142	11810	11686	11775	11601	11637	11649	11619	11662	11710	11702
74	21631	14143	11815	11689	11777	11611	11642	11650	11625	11672	11730	11705
75	21637	14154	11819	11693	11779	11613	11645	11651	11644	11690	11742	11705
76	21647	14160	11828	11702	11782	11619	11658	11657	11669	11710	11742	11711
77	21663	14168	11833	11720	11807	11639	11663	11663	11672	11710	11743	11716
78	21707	14173	11845	11732	11813	11658	11677	11667	11677	11741	11747	11723
79	21709	14184	11852	11734	11818	11678	11691	11683	11678	11742	11748	11736
80	21730	14190	11853	11738	11857	11692	11724	11684	11696	11755	11749	11748
81	21732	14192	11855	11738	11869	11708	11727	11685	11703	11762	11758	11749
82	21750	14236	11856	11745	11877	11731	11735	11708	11704	11762	11770	11755
83	21759	14239	11863	11747	11883	11736	11742	11729	11711	11777	11770	11774
84	21762	14245	11865	11755	11891	11742	11763	11778	11711	11785	11770	11781
85	21791	14247	11882	11791	11896	11745	11785	11785	11720	11786	11790	11797
86	21815	14280	11904	11813	11911	11753	11786	11788	11745	11796	11791	11811
87	21842	14282	11911	11819	11921	11780	11791	11791	11753	11797	11792	11826
88	21850	14290	11914	11819	11971	11789	11793	11791	11767	11814	11813	11844
89	21878	14296	11915	11848	11981	11819	11794	11791	11782	11826	11815	11848
90	21967	14305	11932	11852	12031	11829	11838	11793	11784	11851	11823	11852
91	22033	14362	11981	11914	12081	11842	11839	11802	11813	11853	11825	11865
92	22245	14368	11982	11941	12161	11856	11843	11810	11864	11863	11897	11915
93	22379	14381	11993	11981	12209	11897	11860	11829	11914	11905	11925	11956
94	23981	14381	12001	12005	12226	11959	11862	11850	11916	11912	11991	11982
95	24131	14445	12012	12094	12234	11970	11866	11856	11926	11912	12021	12004
96	24872	14508	12018	12164	12366	12204	11868	11868	11928	11966	12202	12015
97	25920	14584	12048	12168	13028	12299	11922	11890	12027	12060	12274	12127
98	26671	14657	12121	12169	13226	12309	11940	11950	12037	12061	12352	12438
99	29360	14723	12146	12345	14402	12443	11986	11966	12066	12079	12698	13963
100	29678	14772	12571	12403	15923	14426	12137	12004	12091	12114	13062	14493

Uporządkowane rosnąco czasy obliczania asynchronicznego [ms] dla macierzy rozmiaru 1600

Lp.	Wątków											
	1	2	3	4	5	6	7	8	9	10	11	12
1	5370	3037	2890	2944	6848	14982	19441	24827	31889	33374	43689	41532
2	5384	3068	2893	2946	7918	15268	22864	30369	41721	40702	49432	60210
3	5395	3088	2910	2951	8366	15547	23412	31862	47068	44813	51961	63648
4	5406	3092	2914	2978	8402	15634	23557	34615	48178	47366	55717	65152
5	5406	3150	2935	2982	8678	16055	27292	35498	48729	51234	56076	67216
6	5424	3160	2940	2982	8749	16363	27341	36692	50386	51612	65862	70699
7	5446	3332	2941	2991	8955	16678	30433	37938	50808	52476	70701	70861
8	5449	3501	2984	3064	8978	16733	31813	38812	54880	57287	73446	71026
9	5455	3523	3023	3501	9122	16933	32240	39209	57576	58102	75176	83705
10	9096	3695	3613	3590	9527	19150	32271	45819	58171	69688	76361	94893

Uporządkowane rosnąco czasy obliczania asynchronicznego [ms] dla macierzy rozmiaru 2400

Lp.	Wątków											
	1	2	3	4	5	6	7	8	9	10	11	12
1	17762	10187	9387	9423	14593	23999	35790	48039	60394	83316	80349	125517
2	17762	10229	9400	9424	16244	24325	35848	49065	69364	97955	98220	127061
3	17766	10304	9408	9428	16534	24567	36686	54730	73320	97968	101023	127357
4	17775	10342	9408	9435	16622	25796	40764	55093	73975	98024	101110	128370
5	17790	10349	9463	9455	16624	26159	40940	56381	74645	102488	106441	128714
6	17868	10364	9518	9465	16677	26572	41513	56906	75646	103859	112031	132660
7	18004	10418	9540	9481	16873	26931	42574	58084	75758	105465	114286	133447
8	18018	10427	9640	9495	17032	27314	43943	58746	79274	106781	115524	136498
9	18234	10704	9789	9534	17546	28221	44376	60091	79818	108051	127587	142579
10	18302	10890	9923	10030	17879	28861	46788	64275	81098	111547	128275	149194

Uporządkowane rosnąco czasy obliczania asynchronicznego [ms] dla macierzy rozmiaru 3200

Lp.	Wątków											
	1	2	3	4	5	6	7	8	9	10	11	12
1	41766	24005	21765	21751	28454	36594	52121	66957	80272	120340	133332	160400
2	41838	24230	22062	21863	28511	37708	54224	68291	85342	122209	139063	160779
3	41899	24464	22160	21889	29071	37839	55766	69571	89538	122277	145503	162275
4	41939	24469	22300	22104	29234	37964	56015	71342	94508	122474	149892	168162
5	42000	24858	22852	22257	29265	38121	58209	72265	95844	125961	152353	170910
6	42004	28826	25606	22360	29816	38121	59012	73624	101080	129909	155497	177172
7	42050	29452	25675	22590	30070	39082	60360	75524	101613	131773	159125	189147
8	42133	30192	25778	22667	30356	39807	60641	75533	101615	136231	160311	193428
9	42296	32354	25807	22723	30359	42282	60980	76795	108787	139181	160742	193629
10	42662	33720	26047	23457	30828	47310	61301	78345	112986	141178	170312	194649

Uporządkowane rosnąco czasy obliczania asynchronicznego [ms] dla macierzy rozmiaru 4000

Lp.	Wątków											
	1	2	3	4	5	6	7	8	9	10	11	12
1	84555	48649	43416	42362	49683	58125	69768	92507				
2	84560	48692	43432	42932	49838	58772	72445	93732				
3	84611	48889	43485	43206	49924	59237	73040	94739				
4	84815	48928	43542	43226	50061	59800	77965	94863				
5	84858	49025	43635	43327	50264	60350	79020	95380				
6	84872	49217	43769	43347	50762	60400	79495	97630				
7	85276	49396	43825	43407	51003	60736	80140	101964				
8	85503	49573	43988	43778	51332	61013	83188	102896				
9	85588	51475	44171	43914	51375	61488	84047	104406				
10	85615	54009	44358	44010	51683	61867	84592	119129				