PCPC Project

Lucio Squitieri matricola 0522500984

Project: Shelling segregation model

EC2 machine: m4.xlarge

Problema

A cavallo tra gli anni 60 e 70 del Novecento, Schelling condusse degli studi oggetto di pubblicazione nel 1969 e il 1971 con cui si proponeva di indagare l'influenza delle preferenze individuali nel determinare la segregazione spaziale; per far questo, Schelling utilizzò un modello a più agenti intelligenti: a interagire nel sistema erano automi cellulari costituiti da pedine di diverso colore su una scacchiera, il cui movimento da una casella all'altra era condizionato, ogni volta, dall'"**infelicità**" della posizione occupata, a sua volta legato al colore delle pedine più vicine: tali modelli hanno mostrato che è sufficiente che le persone coltivino una blanda preferenza di qualche tipo (ad esempio, etnica, sociale, culturale, ecc.) perché l'effetto di scelte individuali ispirate da tali preferenze debolissime si componga in un fenomeno complessivo di totale segregazione, senza che, nella spiegazione dei fenomeni di separazione in gruppi così nettamente separati, sia possibile distinguere i motivi intenzionali da quelli involontari.

Descrizione della soluzione

Nel caso di questo progetto il sistema o lo spazio è rappresentato da una matrice.

- I tipi di agenti sono due rosso (R) e blu (B) con la presenza di celle vuote.
- Per i due agenti si calcola la soddisfazione e se questa è sotto una certa soglia (in questo progetto è definita al 30%) si deve spostare in un'altra cella.
- Il calcolo della soddisfazione si basa sul controllare quanti agenti dello stesso tipo circondano l'agente analizzato al momento. Nel calcolo della soddisfazioni influisce anche la presenza di celle vuote poichè si suppone che nessuno agente voglia essere rimanere isolato dai suoi simili.

Calcolo della soddisfazione

(100 / vicini) * simili

Se il risultato è superiore a 30 l'agente è soddisfatto.

Per la soluzione bisogna premettere che si usa un modello root-slave indi per cui il processo root inizializza una matrice con agenti casuali e individua tutti i posti vuoti in cui si potrebbero spostare gli agenti insoddisfatti con la funzione:

```
populateInitialMatrix(data, emptyPlaces, N, M);
```

Si ha quindi un array di posizioni elegibili per spostare una agente "emptyPlaces". Questo array viene diviso ai vari processi equamente per permettere a quanti più processi di spostare agenti insoddisfatti (Per restare fedeli alla traccia e quindi permettere uno spostamento casuale, questo array viene mischiato).

Dopo ciò si esegue una scatter della matrice in cui tutti i processi riceveranno tutte le righe necessarie per eseguire il calcolo della soddisfazione. In entrambi i casi viene eseguita una **ScatterV**

Esempio: matrice 3x3 con 3 processi Matrice iniziale:

1 1 1
2 2 2
3 3 3

Matrice ricevuta dal Processo 1:

1 1 1
2 2 2

Matrice ricevuta dal Processo 2:

1 1 1
2 2 2
3 3 3

Matrice ricevuta dal Processo 3:

2 2 2
3 3 3

Si arriva ad un punto quindi in cui **ogni processo ha la sua sottomatrice utilizzabile senza altre informazioni necessarie per il calcolo della soddisfazione** e un array contente la posizione dei k posti liberi che può usare per spostare gli agenti.

Una volta che ogni processo ha la sua sottomatrice si esegue il calcolo della soddisfazione per gli agenti interessati dal processo (evidenziati in rosso) e si inseriscono in un array (se si hanno ancora posti liberi usabili, quindi se k>0). Le posizioni di questi agenti vengono sovrascritte con Empty poiche questi agenti dovranno spostarsi e lasceranno uno spazio vuoto al loro posto. La seguente funzione esegue quanto descritto sopra, è molto complessa poichè va tenuto conto di vari casi per il calcolo della soddisfazione; in particolare i 3 casi in cui si sta analizzando:

- il primo processo e quindi non si hanno righe superiori per quanto concerne la prima riga
- L'ultimo processo e quindi non si hanno righe inferiori all'uultima

- I processi intermedi.

Oltre questi casi bisonga di tenere conto della presenza o meno di elementi precedenti o successivi per eseguire un corretto calcolo della soffisfazione.

```
int getUnsatisfiedAgents(int rank, int rows, char *received_matrix, char *unsatisfied, char *t
    int uns = 0;
    if (rank == 0) {
        for (int i = 0; i < rows - 1; i++) {
            int simili = 0;
            int vicini = 0;
            for (int j = 0; j < M; j++) {
                if (received_matrix[(i * M) + j] != ' ') {
                    if (i > 0) {  // TODO SE HO UNA RIGA SOPRA
                        vicini += 2;
                        if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M) + j]
                        if (received_matrix[(i * M) + j] == received_matrix[(i * M) + j]) simi
                        //controllo se ci sono elementi a sinistra
                        if (j == 0) {  //CI SONO ELEMENTI SOLO A DESTRA
                            vicini += 3;
                            if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M)
                            if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M)
                            if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j +
                        } else if (j == M - 1) {  //CI SONO ELEMENTI SOLO A SINISTRA
                            vicini += 3;
                            if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M)
                            if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M)
                            if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j +
                        } else {
                            vicini += 6;
                            if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M)
                            if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M)
                            if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j -
                            if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M)
                            if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M)
                            if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j +
                        }
                    } else {  //TODO SE NON E' PRESENTE UNA RIGA SUPERIORE
                        vicini += 1;
                        if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M) + j]
                        //controllo se ci sono elementi a sinistra
                        if (j == 0) {  //CI SONO ELEMENTI SOLO A DESTRA
                            vicini += 2;
                            if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M)
                            if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j +
                        } else if (j == M - 1) {  //CI SONO ELEMENTI SOLO A SINISTRA
                            vicini += 2;
                            if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M)
                            if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j -
                        } else {
                            vicini += 4;
                            if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M)
                            if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j -
```

```
                        if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M)
                        if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j +
                    }
                }
            } else
                simili = -1;

            float soddisfazione;
            if (vicini > 0)
                soddisfazione = (100 / vicini) * simili;
            else
                soddisfazione = 100;

            if (soddisfazione < T && received_matrix[(i * M) + j] != ' ') {
                unsatisfied[uns] = received_matrix[(i * M) + j];
                uns++;
                if (liberi > 0) {
                    temp[(i * M) + j] = ' ';
                    liberi--;
                }
            }

            simili = 0;
            vicini = 0;
        }
    }
    return uns;
    //printm(temp, rank, 1);
} else if (rank == world_size - 1) {
    for (int i = 1; i < rows; i++) {
        int simili = 0;
        int vicini = 0;
        for (int j = 0; j < M; j++) {
            if (received_matrix[(i * M) + j] != ' ') {
                if (i != rows - 1) {  //TODO se  ho una riga inferiore
                    vicini += 2;
                    if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M) + j]
                    if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M) + j]
                    if (j == 0) {  //CI SONO ELEMENTI SOLO A DESTRA
                        vicini += 3;
                        if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M)
                        if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M)
                        if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j +
                    } else if (j == M - 1) {  //CI SONO ELEMENTI SOLO A SINISTRA
                        vicini += 3;
                        if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M)
                        if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M)
                        if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j -
                    } else {
                        vicini += 3;
                        if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M)
                        if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M)
                        if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j -
                        if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M)
```

```
                        if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M)
                        if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j +
                    }
                } else {
                    vicini += 1;
                    if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M) + j]
                    if (j == 0) {  //CI SONO ELEMENTI SOLO A DESTRA
                        vicini += 2;
                        if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M)
                        if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j +
                    } else if (j == M - 1) {  //CI SONO ELEMENTI SOLO A SINISTRA
                        vicini += 2;
                        if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M)
                        if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j -
                    } else {
                        vicini += 4;
                        if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M)
                        if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j -
                        if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M)
                        if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j +
                    }
                }
            } else
                simili = -1;
            float soddisfazione;
            if (vicini > 0)
                soddisfazione = (100 / vicini) * simili;
            else
                soddisfazione = 100;

            if (soddisfazione < T && received_matrix[(i * M) + j] != ' ') {
                unsatisfied[uns] = received_matrix[(i * M) + j];
                uns++;
                if (liberi > 0) {
                    temp[((i - 1) * M) + j] = ' ';
                    liberi--;
                }
            }
            simili = 0;
            vicini = 0;
        }
    }
    //printm(temp, rank, 1);
    return uns;
} else {
    for (int i = 1; i < rows - 1; i++) {
        int simili = 0;
        int vicini = 0;
        for (int j = 0; j < M; j++) {
            if (received_matrix[(i * M) + j] != ' ') {
                vicini += 2;
                if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M) + j]) si
                if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M) + j]) si
                if (j == 0) {  //CI SONO ELEMENTI SOLO A DESTRA
                    vicini += 3;
```

```
                    if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M) + (j
                    if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M) + (j
                    if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j + 1)]
                } else if (j == M - 1) {  //CI SONO ELEMENTI SOLO A SINISTRA
                    vicini += 3;
                    if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M) + (j
                    if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M) + (j
                    if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j - 1)]
                } else {
                    vicini += 6;
                    if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M) + (j
                    if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M) + (j
                    if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j - 1)]
                    if (received_matrix[(i * M) + j] == received_matrix[((i - 1) * M) + (j
                    if (received_matrix[(i * M) + j] == received_matrix[((i + 1) * M) + (j
                    if (received_matrix[(i * M) + j] == received_matrix[(i * M) + (j + 1)]
                }
            } else
                simili = -1;
            float soddisfazione;
            if (vicini > 0)
                soddisfazione = (100 / vicini) * simili;
            else
                soddisfazione = 100;

            if (soddisfazione < T && received_matrix[(i * M) + j] != ' ') {
                unsatisfied[uns] = received_matrix[(i * M) + j];
                uns++;
                if (liberi > 0) {
                    temp[((i - 1) * M) + j] = ' ';
                    liberi--;
                }
            }
            vicini = 0;
            simili = 0;
        }
    }
    // printm(temp, rank, 1);
    return uns;
  }
}
```

Si controlla poi se ci sono agenti insoddisfatti e in caso affermativo si ha l'ultima parte che consiste nel trovare i processi proprietari delle posizioni libere assegnate ai diversi processi e ogni processo invia agli altri un messaggio per informarli del numero di struct che potrebbe ricevere; e quindi eseguire una Recv() per le struct da ricevere. Grazie a questa struct il processo ricevente saprà in che posizione inserire l'agente e che agente questo sia (se rosso o blu).

Struct:

```
//definition of the struct that defines the agent to move
    typedef struct {
        int newPosition;  //new position of the agent
        int processo;     //process owner of the position
        char tipo;        //type of agent
    } movingAgent;
```

Funzione che verifica la presenza di agenti insoddisfatti:

```
int uns = getUnsatisfiedAgents(rank, rows, received_matrix, unsatisfied, temp, liberi, world_

        int daSpostare = 0;
        movingAgent *agent = malloc(sizeof(movingAgent) * uns);

        //all gather to allow each processor to share the number of unsatisfied agents
        MPI_Allgather(&uns1, 1, MPI_INT, received, 1, MPI_INT,
                      MPI_COMM_WORLD);

        //check for unsatisfied agents
        bool insoddisfatti = false;
        for (int l = 0; l < world_size; l++) {
            if (received[l] > 0) insoddisfatti = true;
        }
```

Funzione per trovare il processo proprietario della posizione libera: si basa sull'uso dei displacement che utilizzando un array corrispondono direttamente alle posizioni contenute da quel processo.

```
while (receive_size > 0 && uns > 0) {
                int processo = -1;
                //find the corrsponding process to the empty location assigned
                for (int j = 1; j < world_size; j++) {
                    if (received_array[daSpostare] < index[j] && received_array[daSpostare] >=
                        processo = j - 1;
                }
                if (processo == -1) {
                    processo = world_size - 1;
                }
```

Funzione di scambio agenti tra i vari processi:

```
        for (int i = 0; i < world_size; i++) {
                movingAgent *moving = malloc(sizeof(movingAgent) * daSpostare);
                int k = 0;
                for (int j = 0; j < daSpostare; j++) {
                    if (agent[j].processo == i) {
                        moving[k].newPosition = agent[j].newPosition;
```

```
                moving[k].processo = agent[j].processo;
                moving[k].tipo = agent[j].tipo;

                //printf("\n  elementi da inviare rank %d tipo:%c, processo:%d, posizi
                k++;
            }
        }
        //invio k per preparare il ricevente
        if (i != rank) {
            MPI_Request request;
            MPI_Isend(&k, 1, MPI_INT, i, 0, MPI_COMM_WORLD, &request);
            if (k > 0) {  //se k>0 eseguo una Isend per l'invio della struttura

                MPI_Isend(moving, k, sendAgent, i, 1, MPI_COMM_WORLD, &request);
            }
        } else {
            if (k > 0) {
                for (int p = 0; p < k; p++) {
                    //printf("INTERESSE posizione che possiedo in locale %d\n", moving
                    int pos = moving[p].newPosition - index[rank];
                    temp[pos] = moving[p].tipo;
                }
            }
        }
    }

    MPI_Status status;
    int buf;
    for (int i = 0; i < world_size; i++) {
        if (i != rank) {
            //Recv of the number of the elements sent
            MPI_Recv(&buf, 1, MPI_INT, i, 0, MPI_COMM_WORLD, &status);
            if (buf > 0) {  //se k>0 recv per la struttura
                movingAgent *received = malloc(sizeof(movingAgent) * buf);
                MPI_Recv(received, buf, sendAgent, i, 1, MPI_COMM_WORLD, &status);
                for (int k = 0; k < buf; k++) {
                    int pos = received[k].newPosition - index[rank];
                    temp[pos] = received[k].tipo;
                }
            }
        }
    }
```

Una volta avvenuto questo scambio di agenti ogni processo si crea una sua sottomatrice temporanea che viene usata successivamente per una GatherV che permetterà al processo root di stampare la matrice (nel caso tutti gli agenti siano soddisfatti o si sia arrivati al numero massimo di iterazioni consentite) o per procedere all'iterazione successiva.

Nell'iterazione successiva invece di inizializzare la matrice si usa direttamente quella ricevuta con la GatherV e su questa matrice si procede come descritto sopra fino a quando gli agenti sono soddisfatti o si è arrivati al numero massimo di iterazioni consentite. Funzioni importanti

CalculateInitial è la funzione usata per ottenere i valori necessari ad eseguire la scatterV della matrice iniziale. Viene eseguita solo una volta all'inizio del programma poiche i valori necessari sono ottenuti in base alle dimensioni della matrice che rimangono inviariate da una iterazione all'altra.

```
void calculateInitial(int world_size, int *rowProcess, int *sendcounts, int divisione, int res
    int sum = 0;  // Sum of counts. Used to calculate displacements
    for (int i = 0; i < world_size; i++) {
        rowProcess[i] = 0;
        if (i == 0) {
            sendcounts[i] = M;
            displs_modifier[i] = 0;
            for (int j = 0; j < divisione; j++) {
                rowProcess[i] = rowProcess[i] + 1;
                sendcounts[i] += M;
                displs_modifier[i] += M;
                if (resto > 0) {
                    rowProcess[i] = rowProcess[i] + 1;
                    displs_modifier[i] += M;
                    sendcounts[i] += M;
                    resto -= M;
                }

                displs[i] = sum;
            }
            displs[i] += sum;
            sum = sum + displs_modifier[i];
        }

        else if (i == world_size - 1) {
            displs_modifier[i] = 0;
            sendcounts[i] = M;
            for (int j = 0; j < divisione; j++) {
                rowProcess[i] = rowProcess[i] + 1;
                sendcounts[i] += M;
                displs_modifier[i] += M;
                if (resto > 0) {
                    rowProcess[i] = rowProcess[i] + 1;
                    displs_modifier[i] += M;
                    sendcounts[i] += M;
                    resto -= M;
                }
                displs[i] = sum - M;
            }
            sum = sum + displs_modifier[i];
        } else {
            displs_modifier[i] = 0;
            sendcounts[i] = 2 * M;
            for (int j = 0; j < divisione; j++) {
                rowProcess[i] = rowProcess[i] + 1;
                sendcounts[i] += M;
                displs_modifier[i] += M;
                if (resto > 0) {
                    rowProcess[i] = rowProcess[i] + 1;
```

```
                displs_modifier[i] += M;
                sendcounts[i] += M;
                resto -= M;
            }
            displs[i] = sum - M;
        }
        sum = sum + displs_modifier[i];
    }
  }
}
```

PopulateInitialMatrix: usata per la creazione della matrice iniziale con agenti randomici e la memorizzazione della posizione delle celle vuote.

```
int populateInitialMatrix(char *data, int *emptyPlaces, int N, int M) {
    int o = 0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            int simili = rand() % 3;
            if (simili == 0)
                data[(i * M) + j] = 'R';
            else if (simili == 1)
                data[(i * M) + j] = 'B';
            else if (simili == 2) {
                data[(i * M) + j] = ' ';
                emptyPlaces[o] = (i * M) + j;
                o++;
            }
        }
    }
    printm(data, 0, N, N, M);
    return o;
}
```

CalculateEmptyPlaces: usata per calcolare i sendcounts e i displacement necessari per la divisione dell'array contenente le posizioni delle celle vuote

```
void calculateEmptyPlaces(int world_size, int o, int *emptySendcounts, int *emptyDispls) {
    int emptyResto = o % world_size;      //remainder of the division for empty places
    int emptyDivisione = o / world_size;  //reult of the division for empty places
    int emptySum = 0;                     // Sum of counts. Used to calculate displacements of
    for (int i = 0; i < world_size; i++) {
        emptySendcounts[i] = emptyDivisione;
        if (emptyResto > 0) {
            emptySendcounts[i] += 1;
            emptyResto--;
        }

        emptyDispls[i] = emptySum;
        emptySum += emptySendcounts[i];
```

```
    }
}
```

Note implementative

Ho scelto di usare un modello root-slave per la possibilità futura di implementare un sistema in cui la matrice viene fornita direttamente al processo root.

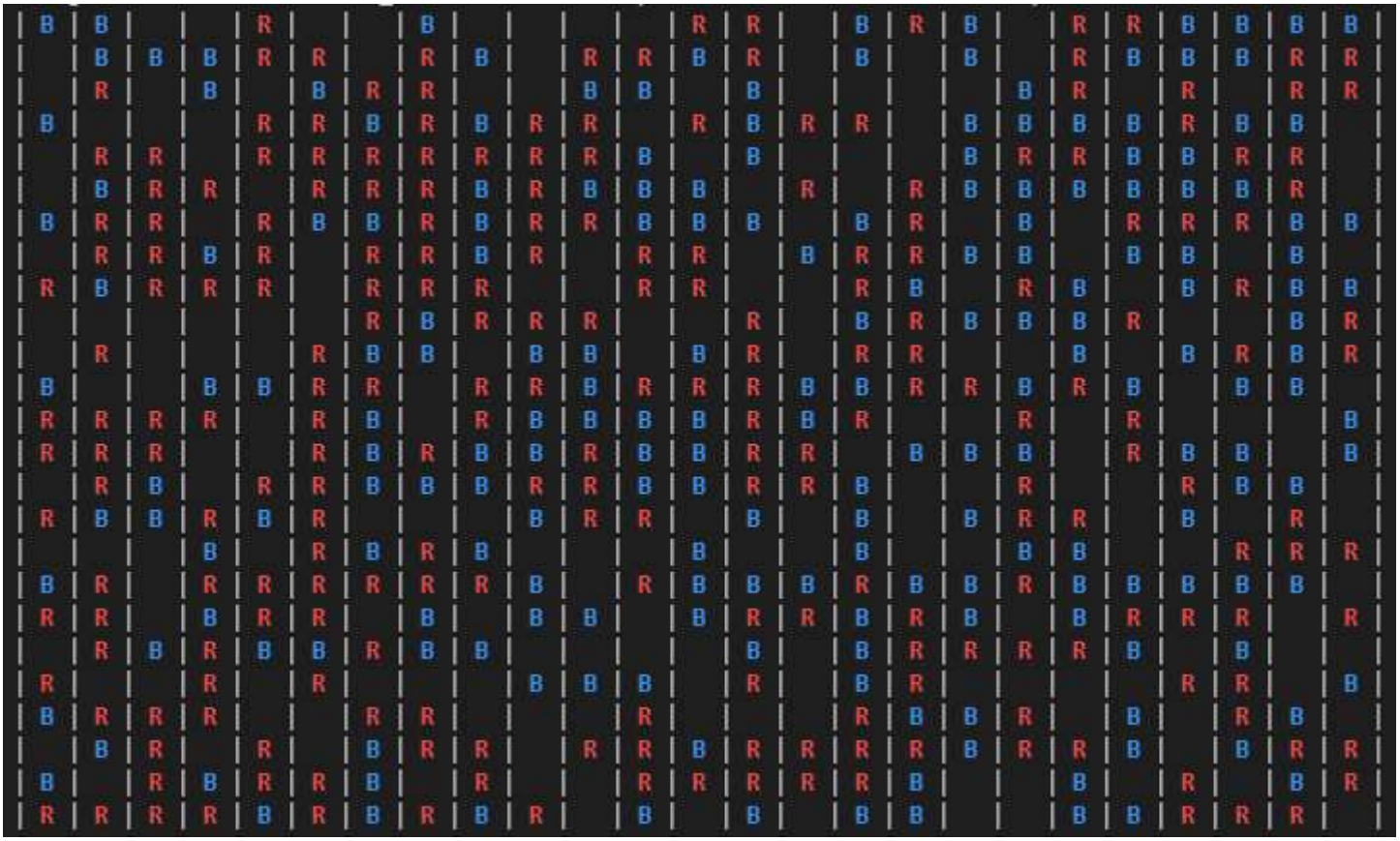
La divisione della matrice è stata eseguita in questo modo per ridurre il più possibile il numero di comunicazioni necessarie che in questo tipo di programmi potrebbe rappresentare un collo di bottiglia per le performance.

Le matrici sono state allocate dinamicamente per permettere l'uso di matrici di grandi dimensioni, cosa impossibile con l'allocazione statica. Ciò ha permesso di passsare le nuove posizione degli agenti sfruttando un solo intero.

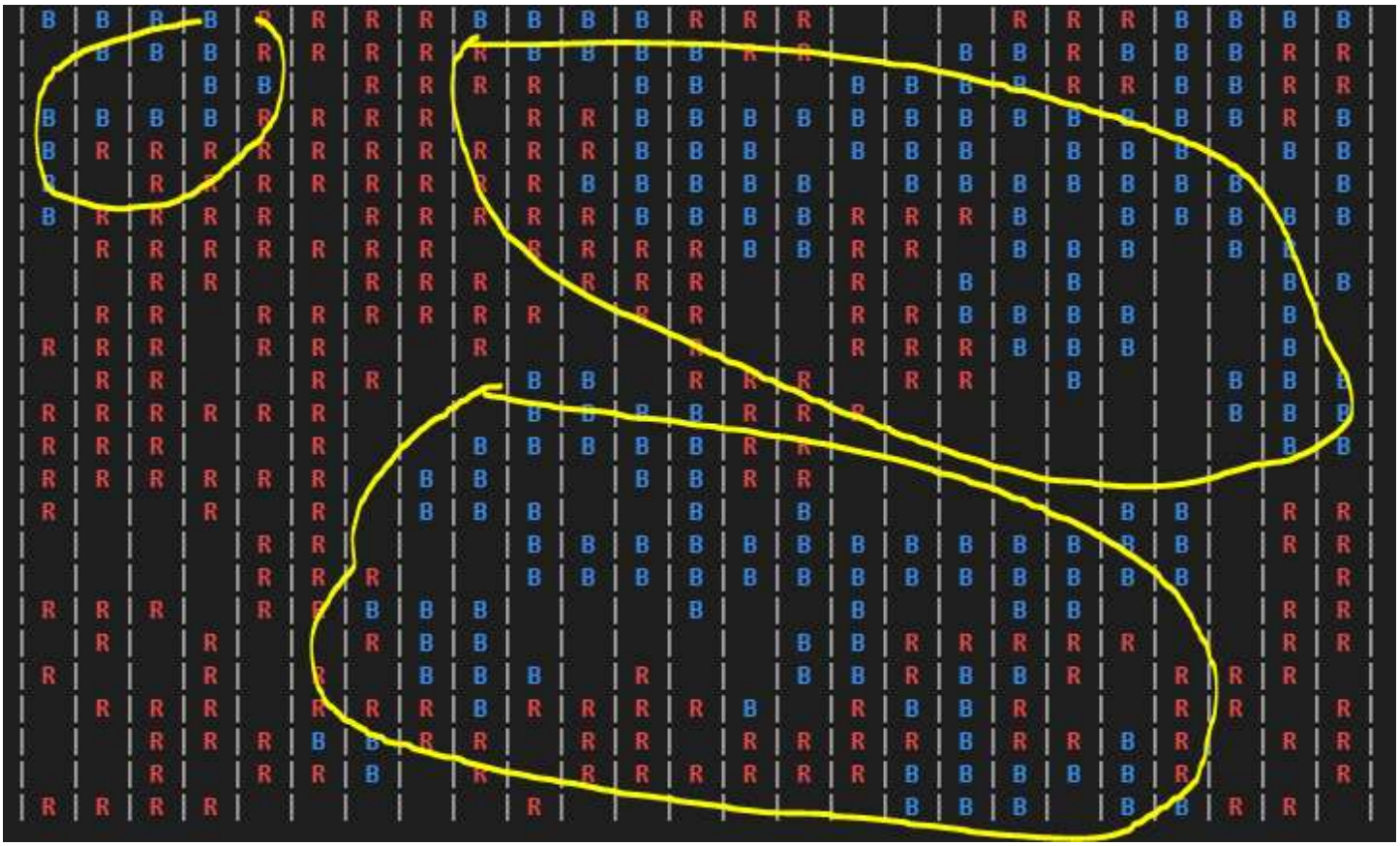
Correctness

La correttezza è difficile da valutare in questo progetto, per mostrarla ho scelto di usare un seed uguale in tutte le iterazioni dei test per la generazione della matrice e per lo shuffle dell'array contentente le posizioni libere. Ottenendo quindi che in tutte le prove sia la matrice iniziale che quella finale sono costanti. Si arriva comunque a notare che alla fine delle iterazioni anche se non tutti gli elementi sono soddisfattti si vengono a creare dei macro gruppi di agenti. Evidenziati nelle figura sottostanti:

Matrice iniziale:



Matrice finale ottenuta tramite gatherv:



Notare che più grande la matrice più viene facile notare la creazione di questi gruppi.

Note per l'esecuzione

usare il comando: mpirun -np “CVPUs” out file “row size” “col size” **Importante** usare un numero di processori minore o uguale del numero di righe altrimenti si possono avere risultati inaspettati. Le stampe sono al momento disabilitate per riabilitarle cercare nel file i commenti "STAMPA" e decommentare la riga interessata. Al momento i seed per la creazione della matrice e dello shuffle dell'array sono uguali se si desidera eseguire tutto randomicamente cercare "SEED" nel file e inserire rand() il seed cche si vuole utilizzare e decommentarlo dove commentato.

Benchmarks

Il benchmark è stato effettuato su un cluser di quattro macchine m4.xlarge per un totale di 16 vCPUs e 64gb di RAM. Sono stati effettuati test per 1, 2, 4, 6, 8, 10, 12, 14, 16 vCPUs con la dimensione della matrice di 1000x1000, 2500x2500, 5000x5000;

La scalabilità forte indica l'accelerazione per una dimensione del problema fissa,infatti è stata testata eseguendo il codice con un differente numero di processori (VCPUs) su una matrice di dimensioni costanti, effettuando diverse rilevazioni di tempo andando poi a calcolare un tempo medio per ogni esecuzione con i diversi processori. L'efficienza della strong scalability è stata calcolata tramite la seguente formula: T1/(P*Tp))*100%

La scalabilità debole indica l'accelerazione per una dimensione variabile del problema al variare del numero di core ed è stata misurata eseguendo il codice con un differente numero di VCPUs e aumentando di 1000 ogni volta il numero di righe per ogni core, tenendo costante il numero di colonne. Anche qui sono state effettuate diverse rilevazioni calcolando poi la media per ogni esecuzione con i diversi processori. L'efficienza della weak scalability è stata calcolata tramite la seguente formula: (T1/Tp)*100%

Lo speedup misura la riduzione del tempo di esecuzione dell’algoritmo eseguito su p processori rispetto all’esecuzione su 1 processore. Lo speed-up in generale è minore del numero di processori mentre lo speed-up ideale assume proprio valore p. Quindi un esecuzione su 2 processori presenta uno speed-up ideale di 2 e cosi via. Lo speed-up per la strong scalability è stato calcolato mediante la seguente formula: T1 /Tp dove T1 è il tempo d'esecuzione su un singolo processore mentre Tp è il tempo d'esecuzione dell'algoritmo con p processori.

Matrice 1000x1000

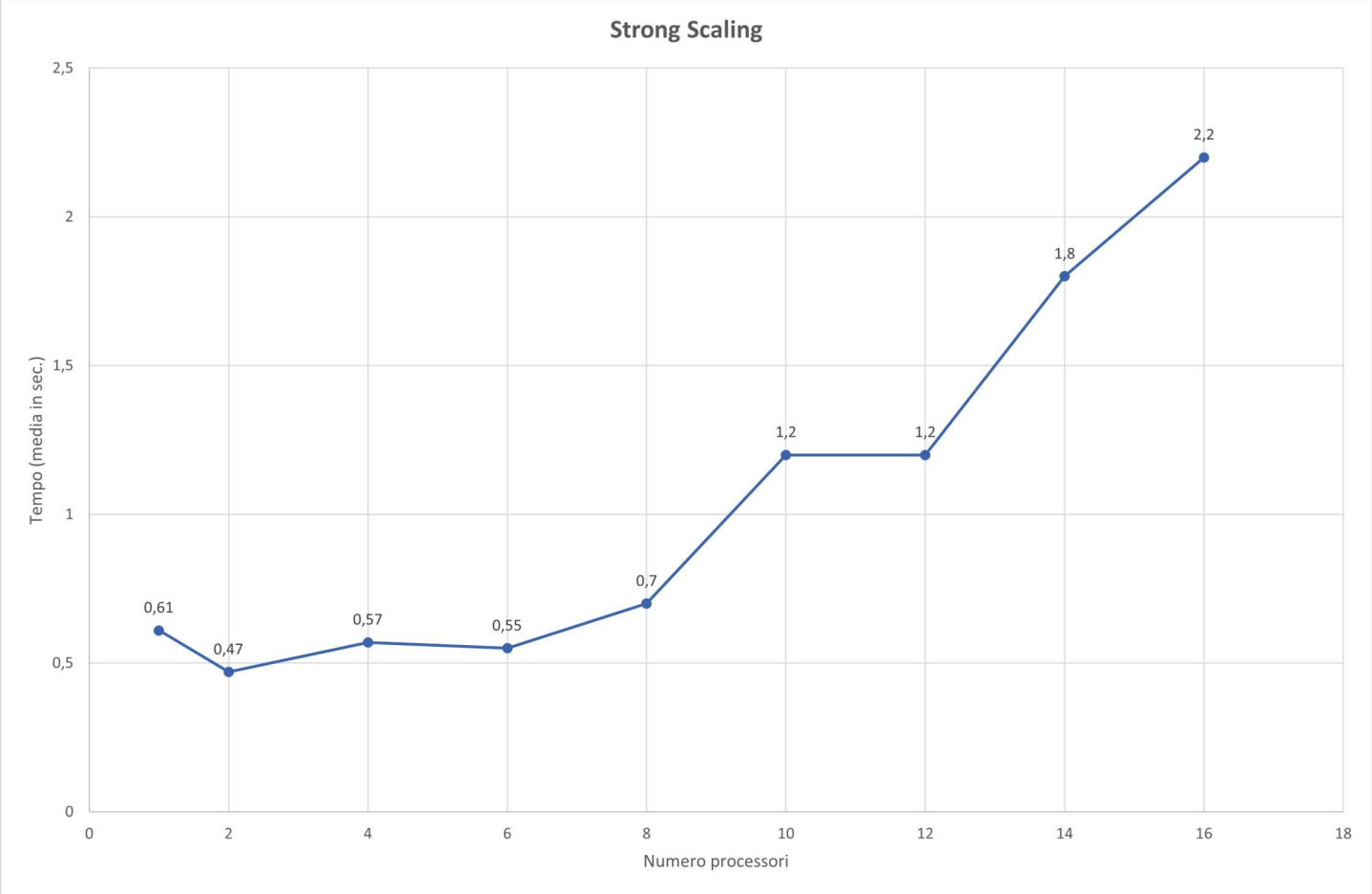
Speedup

VCPUs	1	2	4	6	8	10	12	14	16
Speed-up	1.00	1.29	1.07	1.10	0.87	0.51	0.51	0.34	0.28

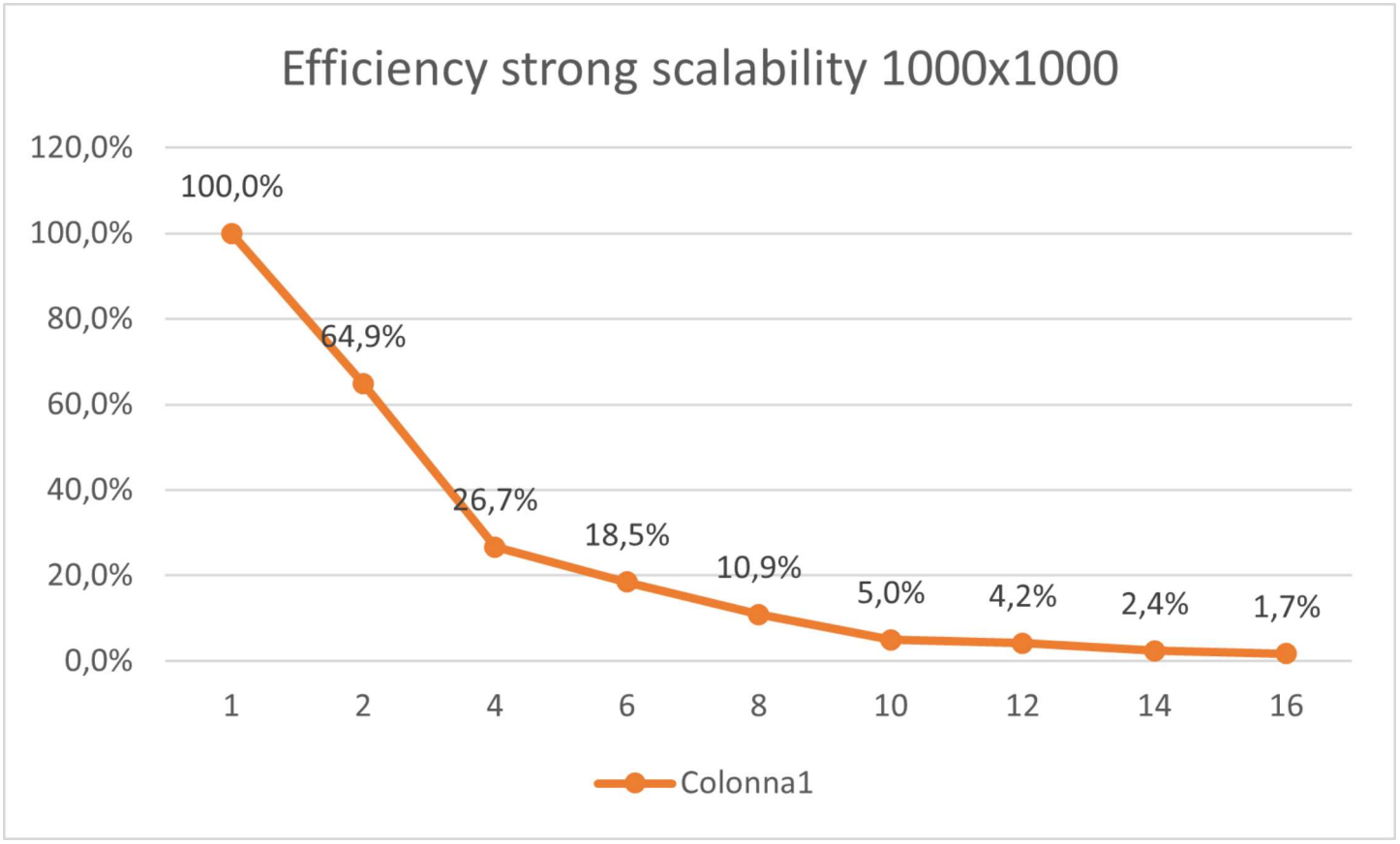
Strong Scalability

VCPUs	1	2	4	6	8	10	12	14	16
Efficienza	100%	64.9%	26.7%	18.5%	10.9%	5%	4.2%	2.4%	1.7%

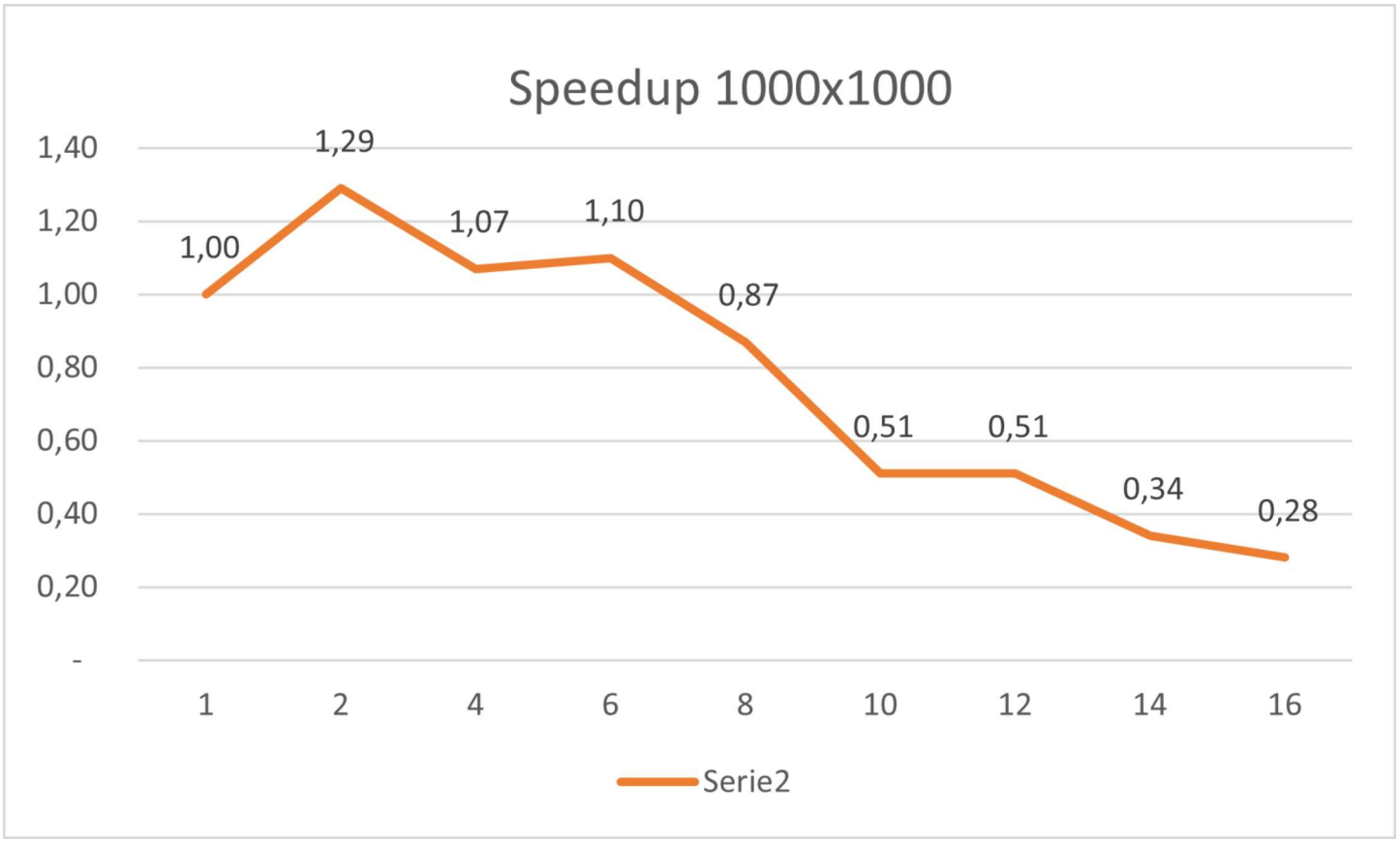
Strong scalability: I valori nel grafico rappresentano il tempo, in secondi, passato per l'esecuzione del programma



Efficienza strong scalability:



Speedup:



Matrice 2500x2500

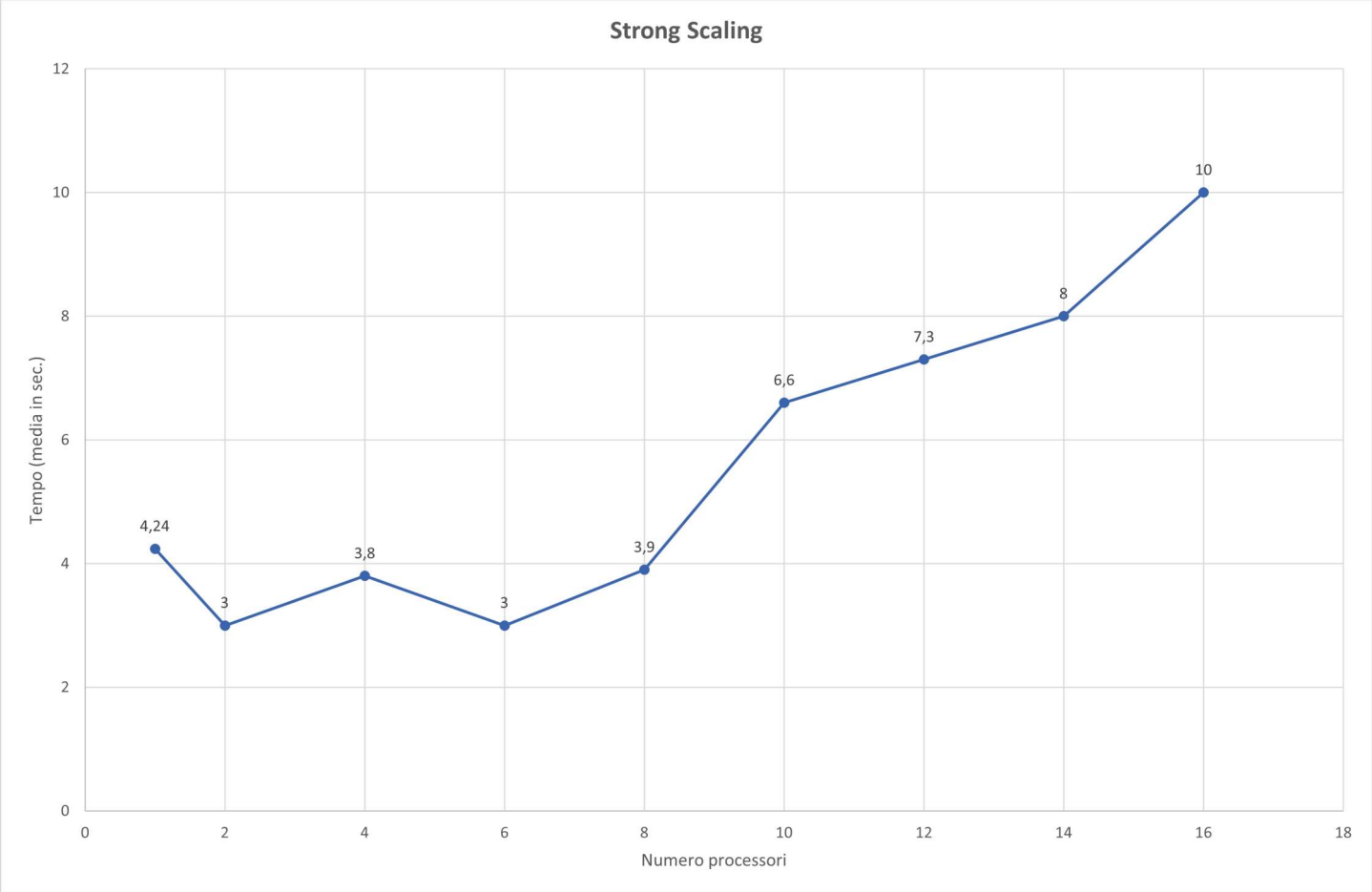
Speedup

VCPUs	1	2	4	6	8	10	12	14	16
Speed-up	1.00	1.41	1.11	1.41	1.10	0.60	0.58	0.53	0.42

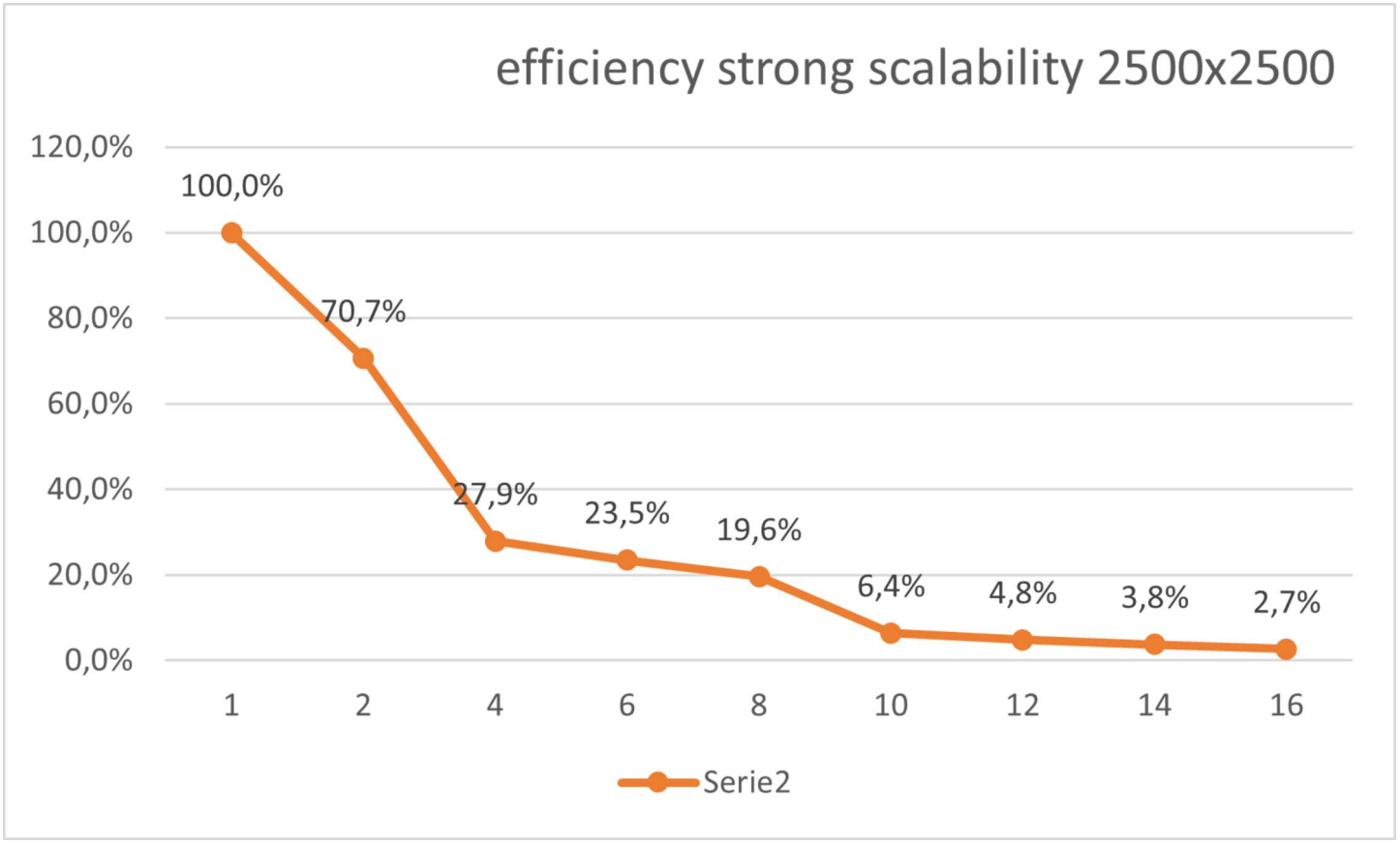
Strong Scalability

VCPUs	1	2	4	6	8	10	12	14	16
Efficienza	100%	70.7%	27.9%	23.5%	19.6%	6.4%	4.8%	3.8%	2.7%

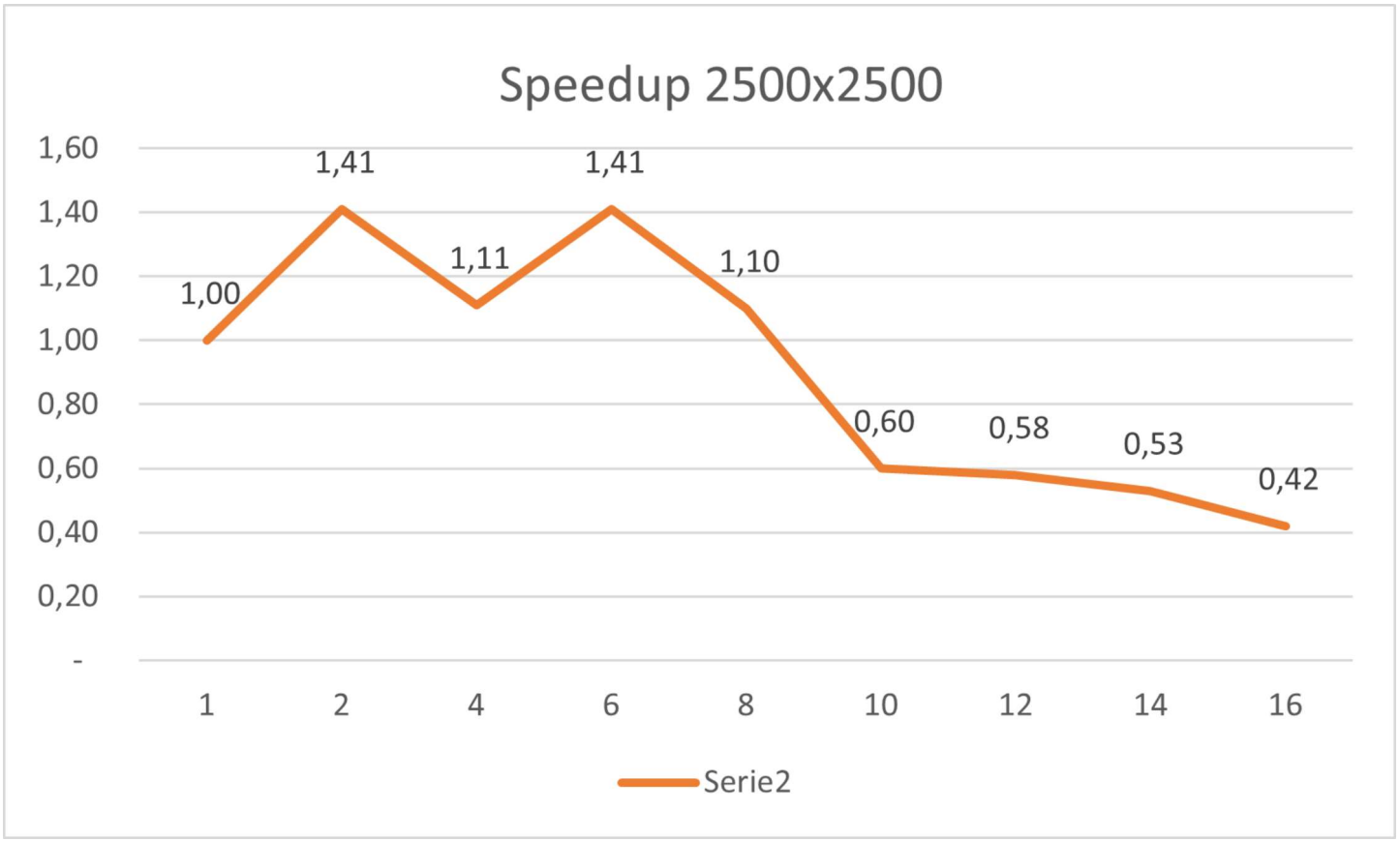
Strong scalability: I valori nel grafico rappresentano il tempo, in secondi, passato per l'esecuzione del programma



Efficienza strong scalability:



Speedup:



Matrice 5000x5000

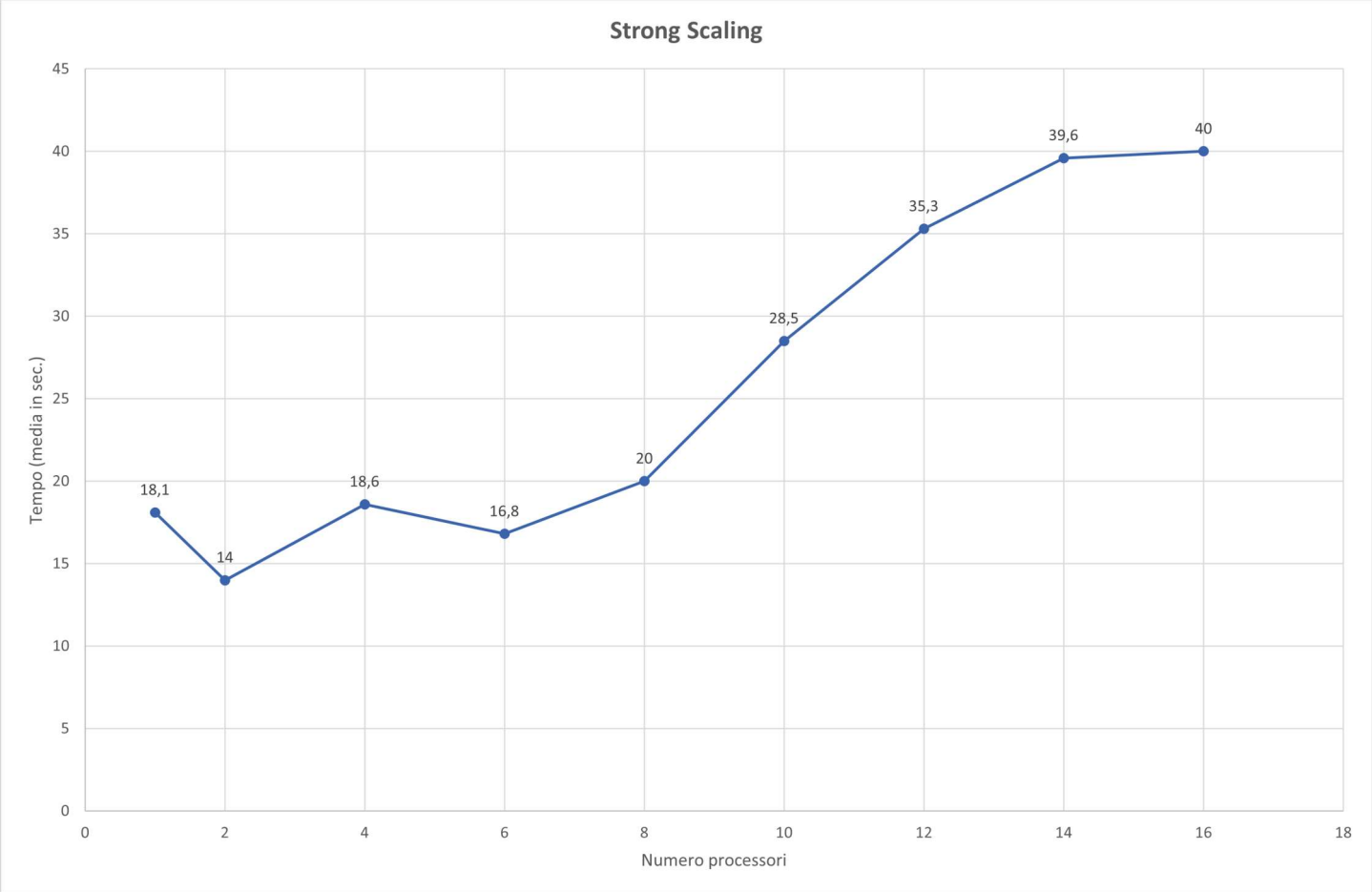
Speedup

VCPUs	1	2	4	6	8	10	12	14	16
Speed-up	1.00	1.29	0.27	0.97	1.07	0.90	0.63	0.51	0.45

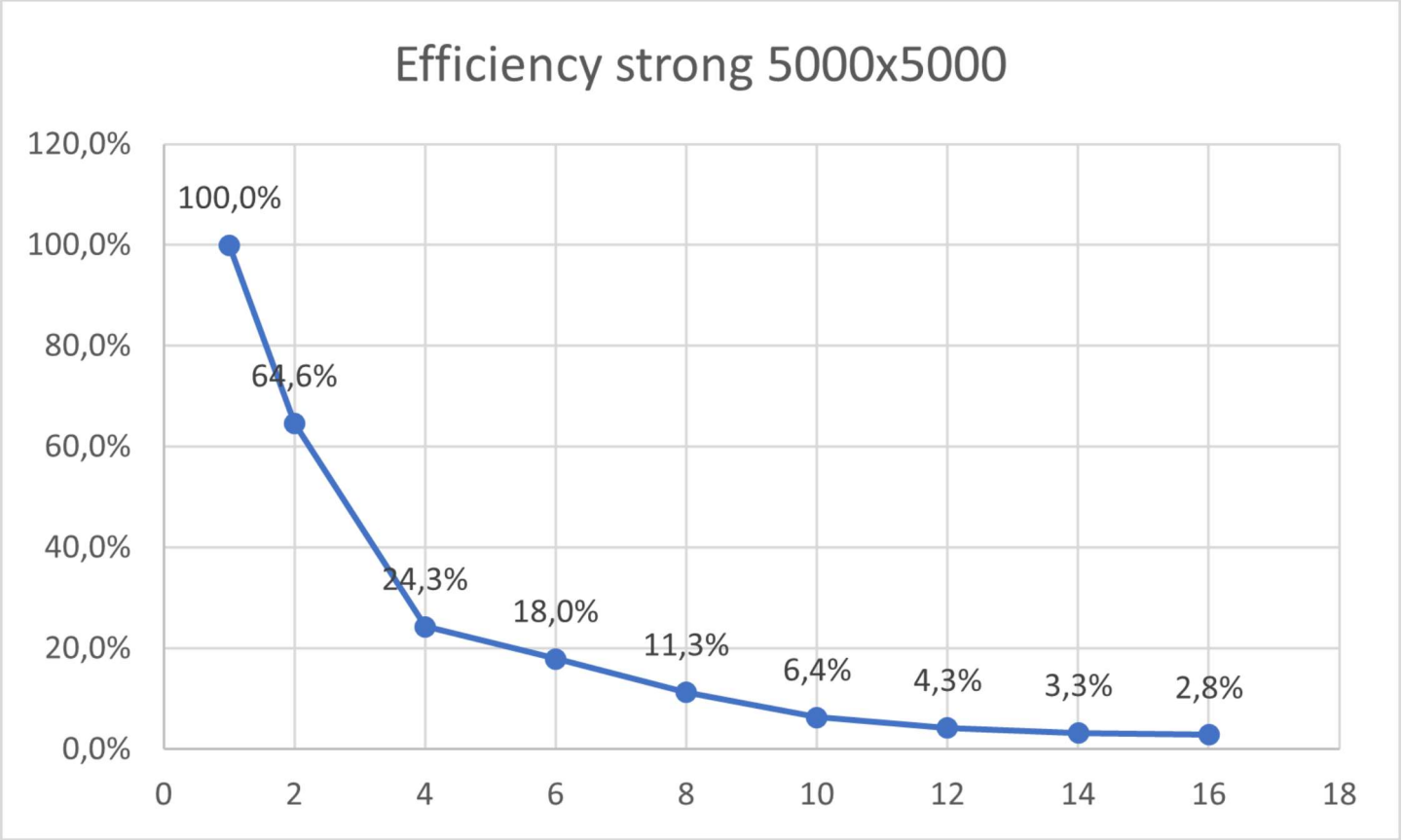
Strong Scalability

VCPUs	1	2	4	6	8	10	12	14	16
Efficienza	100%	64.6%	24.3%	18%	11.3%	6.4%	4.3%	3.3%	2.8%

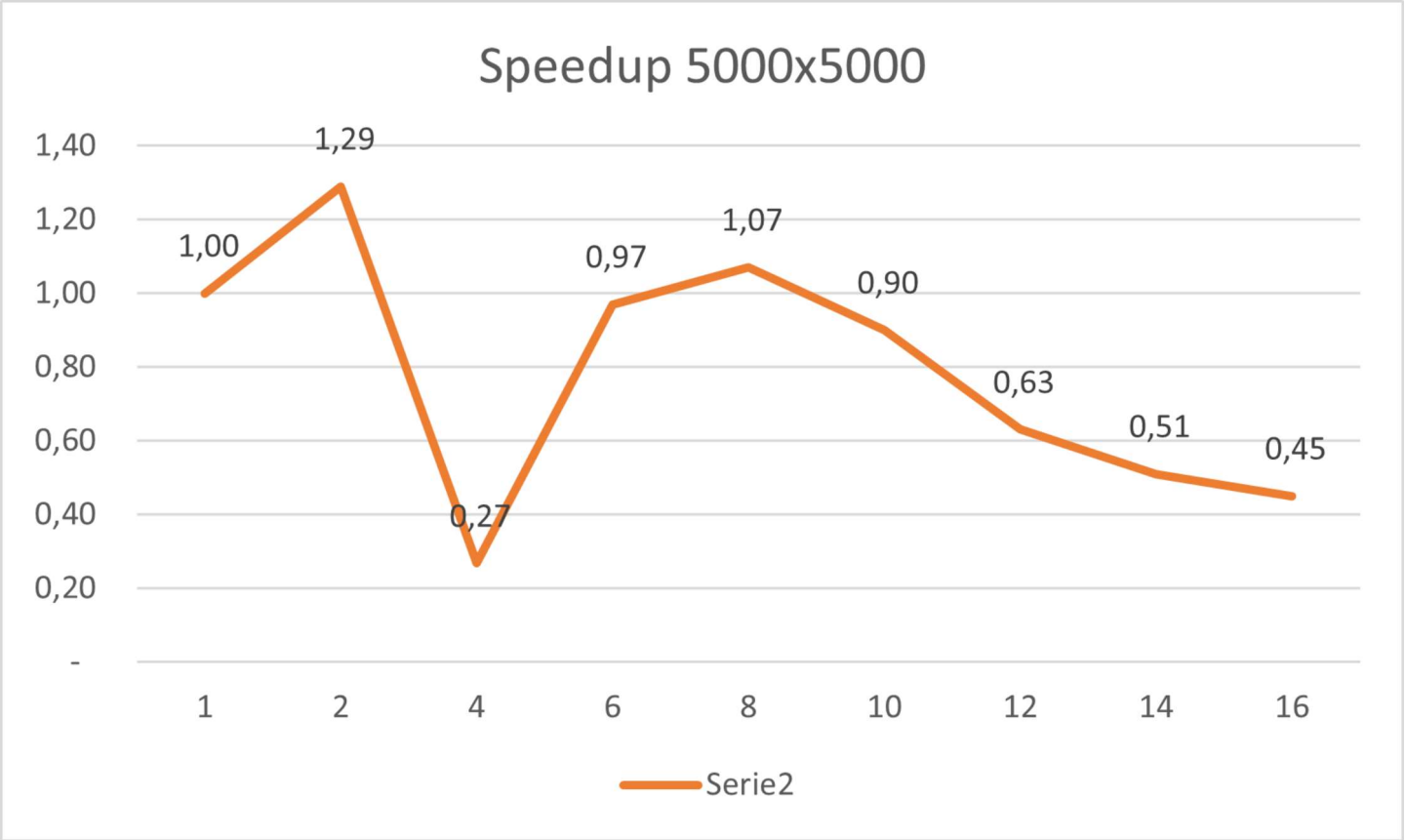
Strong scalability: I valori nel grafico rappresentano il tempo, in secondi, passato per l'esecuzione del programma



Efficienza strong scalability:



Speedup:

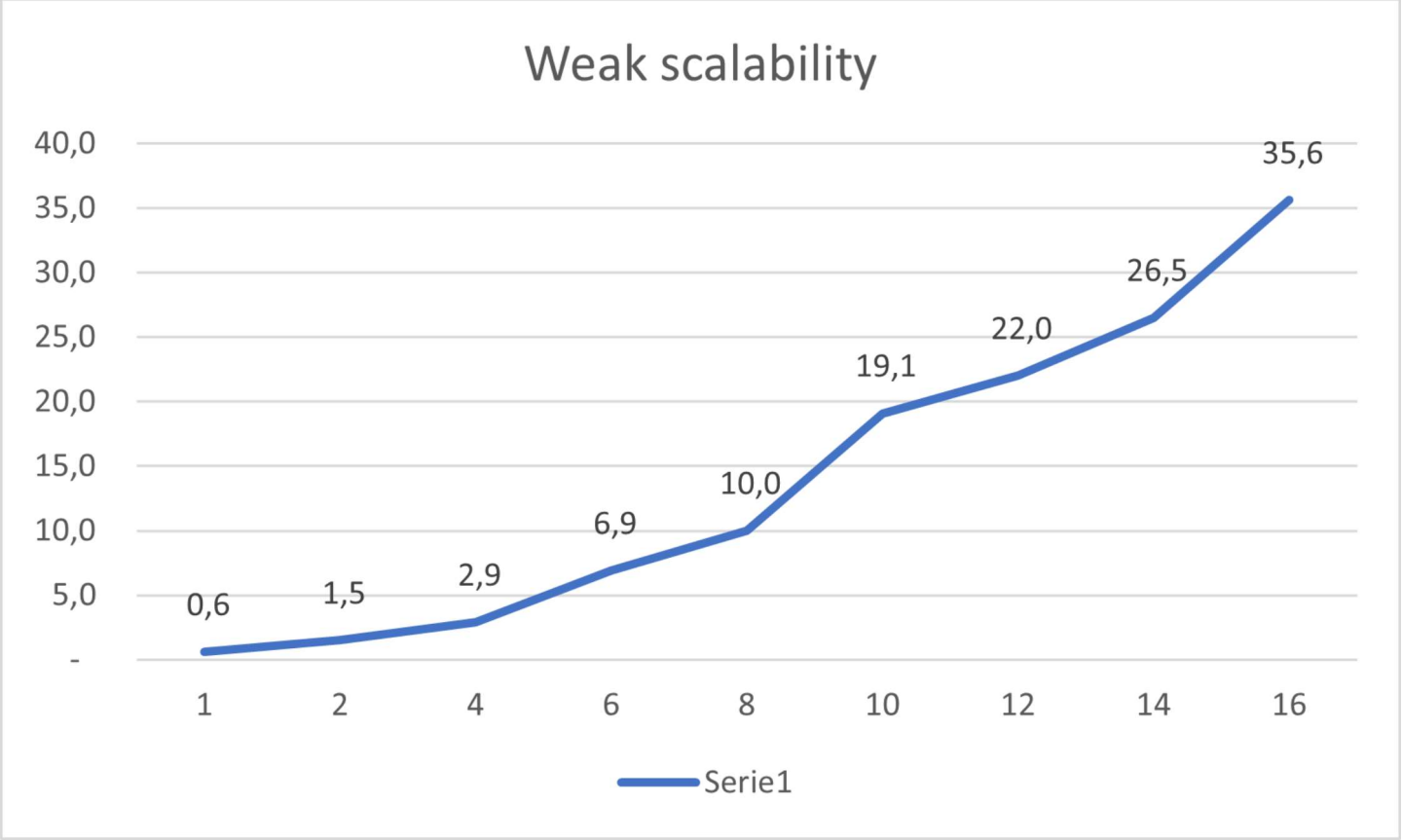


Weak scalability

Strong Scalability

VCPUs	1	2	4	6	8	10	12	14	16
Numero righe	1000	2000	4000	6000	8000	10000	12000	14000	16000

I valori nel grafico rappresentano il tempo, in secondi, passato per l'esecuzione del programma



Risultati

Dopo un'analisi dei tempi e dei dati ricavati da essi si può evincere come l'aumento dei processori non si traduce in uno speedup diretto del programma proprio per via della sua bassa efficienza. Anzi si può notare come sia più efficiente con meno processori e che con 4 e 6 processori si abbiano i risultati migliori. Ipotizzo che la bassa efficienza del programma sia da imputare alla grande mole di dati e di comunicazioni che sono necessari nella mia implementazione.