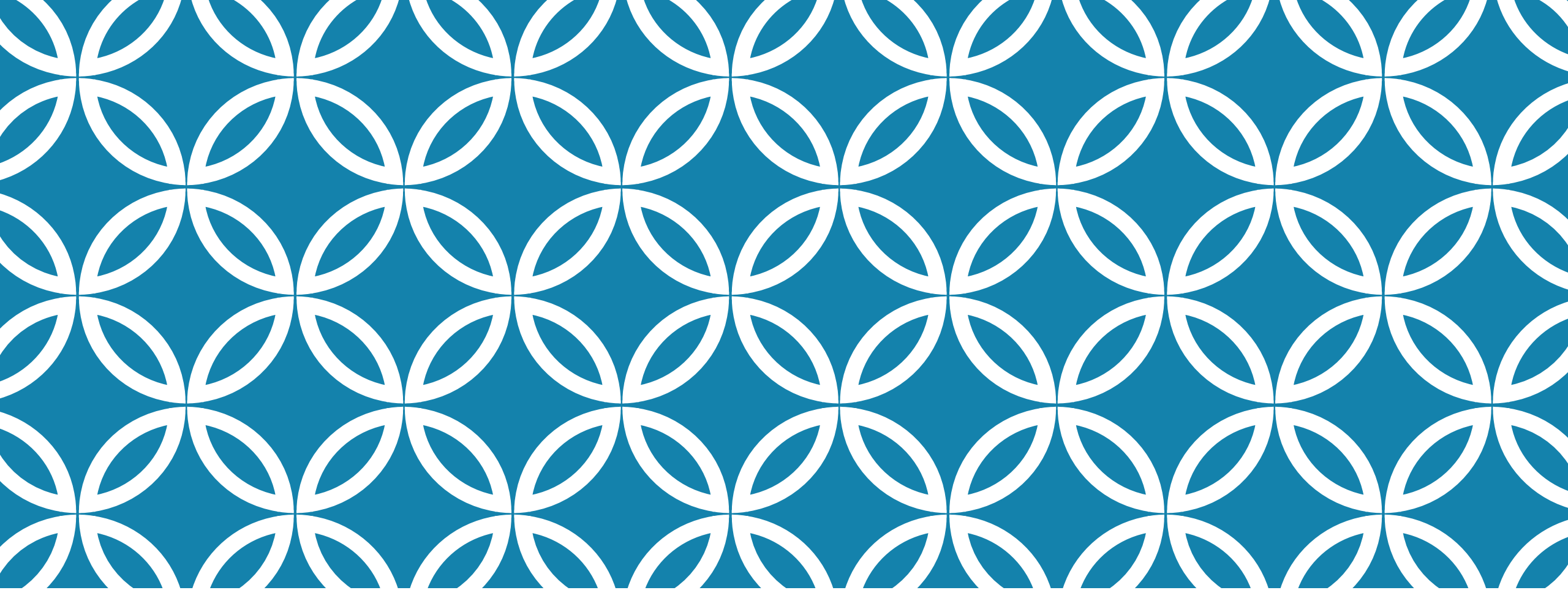


N-GRAMS

RUCI SINDI
7090797
sindi.ruci@stud.unifi.it

COSA SONO GLI N-GRAMMI

Sono porzioni di testo di grandezza n. Ad esempio un bi-gramma è una porzione di testo di grandezza due, nella parola CASA i bigrammi sono ca-as-sa.

CODICE SEQUENZIALE

Il mio codice sequenziale esegue i seguenti passi:

- Legge un file già ripulito da punteggiatura e caratteri speciali
- Calcola gli n-grammi presenti nel file e le loro cardinalità
- Si tiene traccia dei calcoli in una mappa che associa a ogni n-gramma la sua cardinalità

```
my_file = utils.format_text( file_name: "Testo 10MB.txt");

auto start_time_1 :time_point<...> =  high_resolution_clock::now();

dictionary_main = utils.calculate_n_gram( n_gram_dim: nGramDimention,  text: my_file);
```

DETTAGLIO DELLE DUE FUNZIONI PRINCIPALI

```
string format_text(string file_name)
{
    fstream text;
    string my_text;

    text.open( s: file_name,  mode: ios::in);
    if (!text)
    {
        cout << "No such file";
    }
    else
    {

        while (1)
        {
            char ch;
            text >> ch;
            if (text.eof())
                break;
            my_text.push_back( c: ch);
        }
    }

    text.close();

    return my_text;
```

```
map<string, int> calculate_n_gram(int n_gram_dim, string text){
    map<string, int> dictionary;


    for (int i = 0; i < text.size() - n_gram_dim +1; i++)
    {
        string n_gram;

        for (int j = 0;  j < n_gram_dim; j++)
        {
            n_gram.push_back( c: text[ i + j]);
        }

        if (dictionary.count( x: n_gram))
        {
            dictionary[n_gram] = dictionary[n_gram] + 1;
        } else
        {
            dictionary.insert( x: pair<string, int> ( &: n_gram,  y: 1));

        }

    }

    return dictionary;
```

TEMPI DI ESECUZIONE

Ho passato al programma file di lunghezza diversa da analizzare per osservare i tempi di esecuzione, i file impiegati sono delle seguenti grandezze:

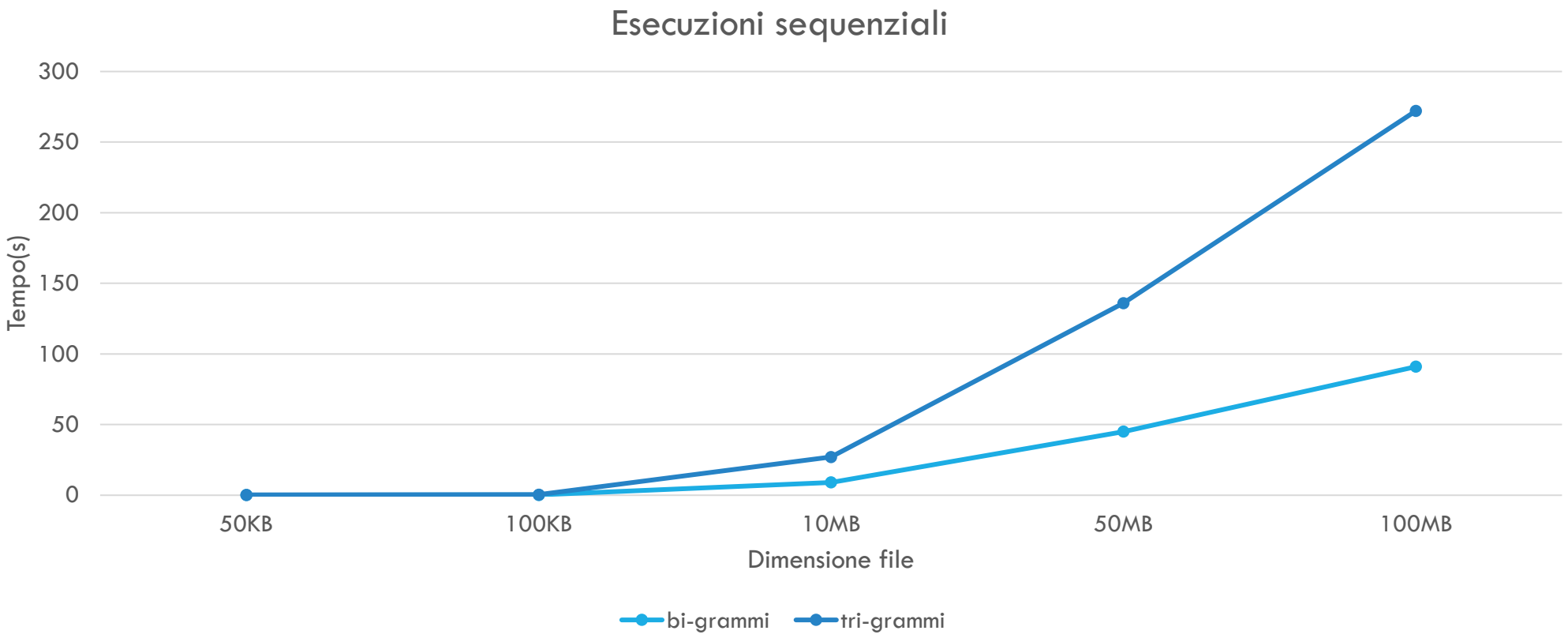
- 50KB
- 100KB
- 10MB
- 50MB
- 100MB

TEMPI DI ESECUZIONE

I tempi risultanti sono stati, rispettivamente bi-grammi e tri-grammi:

•50KB	58millisec	163millisec
•100KB	118millisec	319millisec
•10MB	9sec	27sec
•50MB	45sec	136sec
•100MB	91sec	272sec

GRAFICO TEMPI ESECUZIONE



CODICE PARALLELO

Il codice parallelo segue i seguenti passi:

- Legge un file già ripulito da punteggiatura e caratteri speciali
- Divide il testo in tante parti quanti i thread disponibili
- Su ogni parte di testo esegue l'analisi degli n-grammi come nel codice sequenziale

```
text = par.format_text( file_name: "Testo 10MB.txt");

int k = round( x: text.size()/cores);
auto start_time_2 : time_point<...> =  high_resolution_clock::now();

omp_set_num_threads(cores);
# pragma omp parallel
# pragma omp for
    for (int i = 0; i < cores; i++)
    {
        thread_dictionaries[i] = par.parallel_thread( start: i*k, finish: ((i+1) * k) + (n_gram_dimension - 1) - 1, dimension: n_gram_dimension, text_local: text);
    }
```

DUE FUNZIONI PRINCIPALI DEL CODICE PARALLELO

```
static string format_text(string file_name)
{
    fstream text;
    string my_text;

    text.open( s: file_name,  mode: fstream ::in);
    if (!text)
    {
        cout << "No such file"<< endl;
    }
    else
    {

        while (1)
        {
            char ch;
            text >> ch;
            if (text.eof())
                break;
            my_text.push_back( c: ch);
        }
    }

    text.close();

    return my_text;
}
```

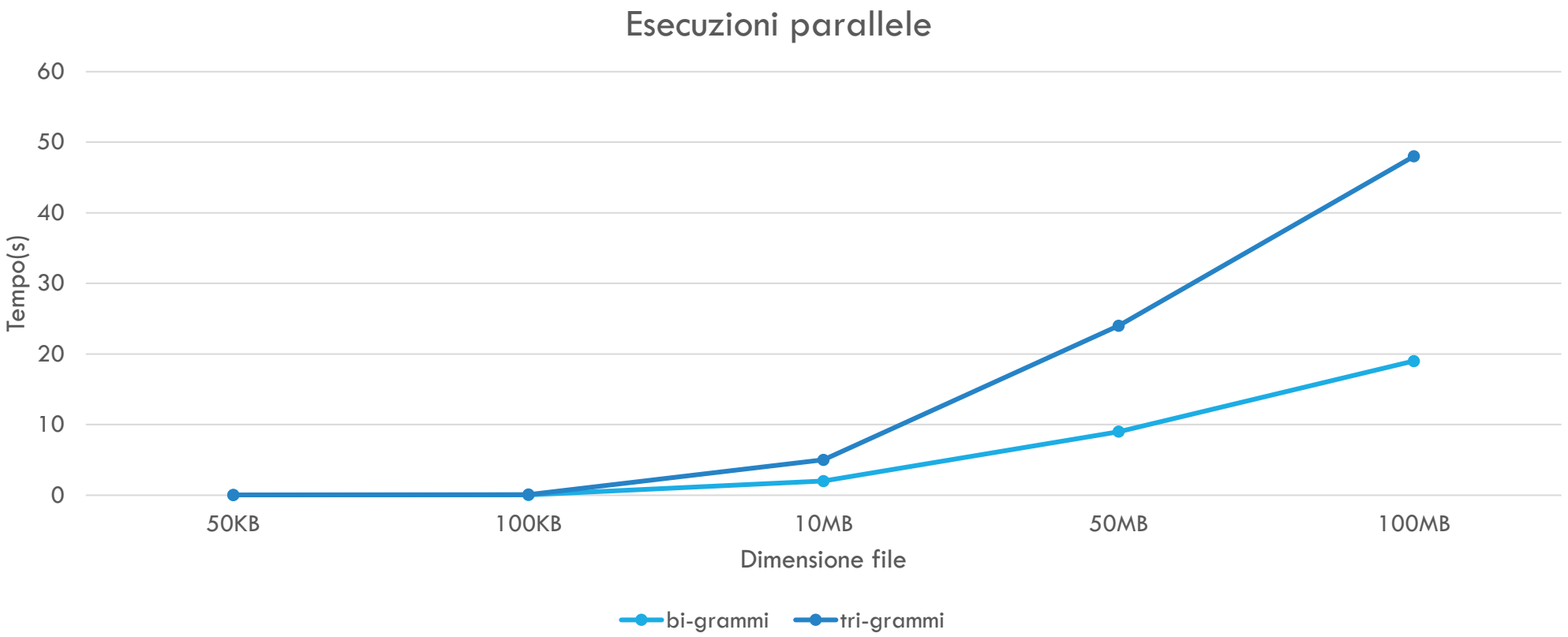
```
map<string, int> parallel_thread(int start, int finish, int dimension, string text_local)
{

map<string, int> local_dictionary;

if (finish > text_local.size())
{
    finish = text_local.size() - 1;
}

for ( int i = start + dimension - 1; i <=  finish; i++)
{
    string n_gram;
    for (int j = dimension - 1; j >= 0; j--)
    {
        n_gram.push_back( c: text_local[i - j]);
    }

    if (local_dictionary.count( x: n_gram))
    {
       local_dictionary[n_gram] = local_dictionary[n_gram] + 1;
    } else
    {
        local_dictionary.insert( x: pair<string, int> ( &: n_gram,  y: 1));

    }

}

return local_dictionary;
```

TEMPI DI ESECUZIONE

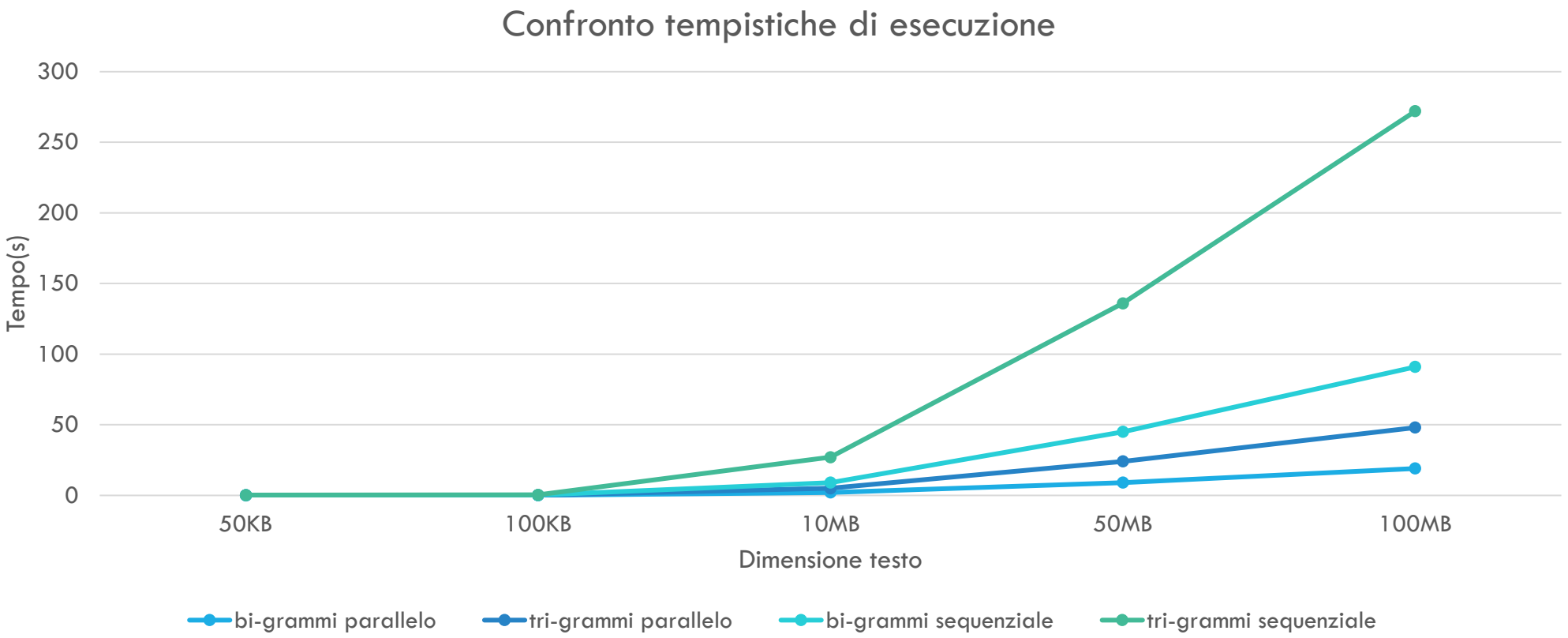
I tempi risultanti sono stati, rispettivamente bi-grammi e tri-grammi:

•50KB	21millisec	36millisec
•100KB	34millisec	78millisec
•10MB	2sec	5sec
•50MB	9sec	24sec
•100MB	19sec	48sec

GRAFICO TEMPI DI ESECUZIONE



SEQUENZIALE VS PARALLELO



CONFRONTO TRA THREAD

Mantenendo fissa la dimensione del file e variando il numero di thread e di n-grammi calcolarti, si ottengono i seguenti risultati

Notiamo come usando più thread di quelli disponibili, le prestazioni calano.