UNIVERSITÀ
DEGLI STUDI
FIRENZE

Bigrams and trigrams

Andrea Simioni

Introduction

The main purpose of the project is compute and estimate occurrences of bigrams and trigrams in a certain text.

More in general:

- A sequence of **two** letters (e.g. of) is called a **bigram**.
- A **three**-letter sequence (e.g. off) is called a **trigram**.
- The general term **n-gram** means '**sequence of length n**'.

n	Letters	Names
1	p	unigram
2	pa	bigram
3	par	trigram

Examples

Bigram	Occurrence
aa	98
ab	56
ac	277

Occurrences of bigrams

Trigram	Occurrence
cce	18
cci	13
cch	12

Occurrences of trigrams

Technologies

- Sequential Implementation: **Java**
- Parallel Implementation: **Java Thread**

Computation

- Sequential version:

Algorithm 1: Sequential version

```
   Data: n, filestring
1  for i = 0 to filestring.length-n+1 do
2  |   key = "";
3  |   for j = 0 to n - 1 do
4  |   |   key = key + filestring[i+j];
5  |   end
6  |   // insert key in a data structure;
7  end
```

- n = n-grams
- filestring = contains the characters from txt file

- Parallel version

Algorithm 2: Parallel version

```
   Data: id, k, n, begin, stop, filestring
1  for i = this.begin to (this.stop - this.n+1) do
2  |   key = "";
3  |   for j = 0 to this.n - 1 do
4  |   |   key = key + filestring[i+j];
5  |   end
6  |   // insert key in a data structure;
7  end
```

- id = idthread
- k = floor(text.length/nThreads) // Dimension of test splits
  where nThread = Number of threads
- n = n-grams
- begin = (k * i)
- stop = (i+1)*k + ((n-1)-1)
- filestring = contains the characters from txt file

Thread's attributes

- id
- begin
- stop
- filestring

Sequential implementation

Data preprocessing

- ***readAndCleanText()***: read the text to analyze and replace opportunely those characters defined *invalid.*

Data structure

- **HashMap**: constant time performance for the basic operation (*get(), put()*) for large sets.
- Data are stored in **(Key, Value)** pairs.

Computation

- ***computeNgrams()***: follows the sequential computation explained above.
  - *int n*: identifies the number of grams (two or three).
  - *char [] filestring*: text to analyze.

Parallel implementation

- Divide the text in as many parts as are the thread instances.
- Entrust the search of bigrams and trigrams on a single part to a single thread.

Example

```
Verso la metà degli anni novanta, Apple si trovò a
fare i conti con un sistema operativo, Mac OS,
che aveva raggiunto i suoi limiti strutturali di
sviluppo. L'architettura a multitasking cooperativo
era ormai una tecnologia sorpassata e quindi, nel
1994, decise di avviare il progetto Copland, con
lo scopo di creare un nuovo sistema operativo
moderno e libero dalle limitazioni tecniche del
precedente Mac OS.Il progetto fallì nel 1996 per
motivazioni tecniche e di politica interna (alcune
migliorie di Copland vennero integrate nel Mac OS 8
rilasciato nel 1997).
```

1 thread

```
Verso la metà degli anni novanta, Apple si trovò a
fare i conti con un sistema operativo, Mac OS,
che aveva raggiunto i suoi limiti strutturali di
sviluppo. L'architettura a multitasking cooperativo
era ormai una tecnologia sorpassata e quindi, nel
1994, decise di avviare il progetto Copland, con
```

```
lo scopo di creare un nuovo sistema operativo
moderno e libero dalle limitazioni tecniche del
precedente Mac OS.Il progetto fallì nel 1996 per
motivazioni tecniche e di politica interna (alcune
migliorie di Copland vennero integrate nel Mac OS 8
rilasciato nel 1997).
```

2 threads

```
Verso la metà degli anni novanta, Apple si trovò a
fare i conti con un sistema operativo, Mac OS,
che aveva raggiunto i suoi limiti strutturali di
sviluppo. L'architettura a multitasking cooperativo
```

```
era ormai una tecnologia sorpassata e quindi, nel
1994, decise di avviare il progetto Copland, con
lo scopo di creare un nuovo sistema operativo
moderno e libero dalle limitazioni tecniche del
```

```
precedente Mac OS.Il progetto fallì nel 1996 per
motivazioni tecniche e di politica interna (alcune
migliorie di Copland vennero integrate nel Mac OS 8
rilasciato nel 1997).
```

3 threads

Parallel implementation - Java thread

Some features of *java.util.concurrent* package have been used:

- **Future**
- **ExecutorService**

Data preprocessing

- ***readAndCleanText()***

Data structure

- **ConcurrentHashMap:**
  - Allows concurrent modifications of the map from several threads without blocking them.
  - It’s lock-free on reads.
  - Lock only the bucket that’s being altered.
  - Data are stored in **(Key, Value)** pairs.

Parallel implementation - Java thread

Computation

- **Declare** a thread class which implements *callable.*
- **Implement** the *call()* method which computes bigrams and trigrams as described before.
- **Implement** a *Merge()* function to merge the maps returned from threads.
- **Instantiate** a Future array and an ExecutorService specifying the thread pool size.
- **Use** the ExecutorService object to submit the compute method and get the results through *.get()* Future method.

Parallel implementation - different approaches

Second Approach:

- Merge function is computed directly in the thread itself, instead of being processed externally.
- The final **concurrentHashmap** is passed as a parameter when the thread is instantiated.
- Thread uses a normal **HashMap** for store occurrences of each n-grams on the part of text assigned.
- instead of returning the value, it merges directly the result with the final **concurrentHashmap.**
- The Thread Class implements *Runnable* and no more *Callable,* because the thread hasn't to return any value.
- A further consequence of this is that the feature *Future* has not been used in this version.

Third Approach:

- Hybrid between the two previous versions.
- The implementation is almost identical to the first approach, with the only difference that thread uses a normal **HashMap** for store occurrences of each n-grams on the part of text assigned.

Tests

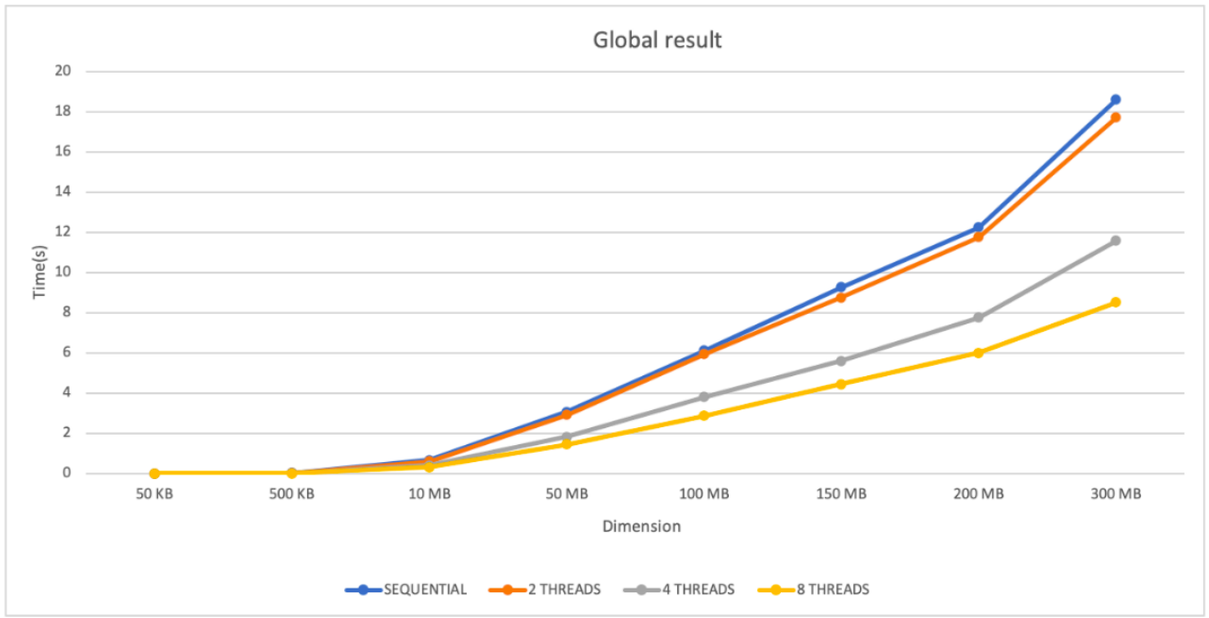
Sequential version:

- The computational time has been collected varying the text dimension (50KB, 500KB, 10MB, 50MB, 100MB,150MB, 200MB and 300MB).
- Every result is calculated on the average of 10 runs.

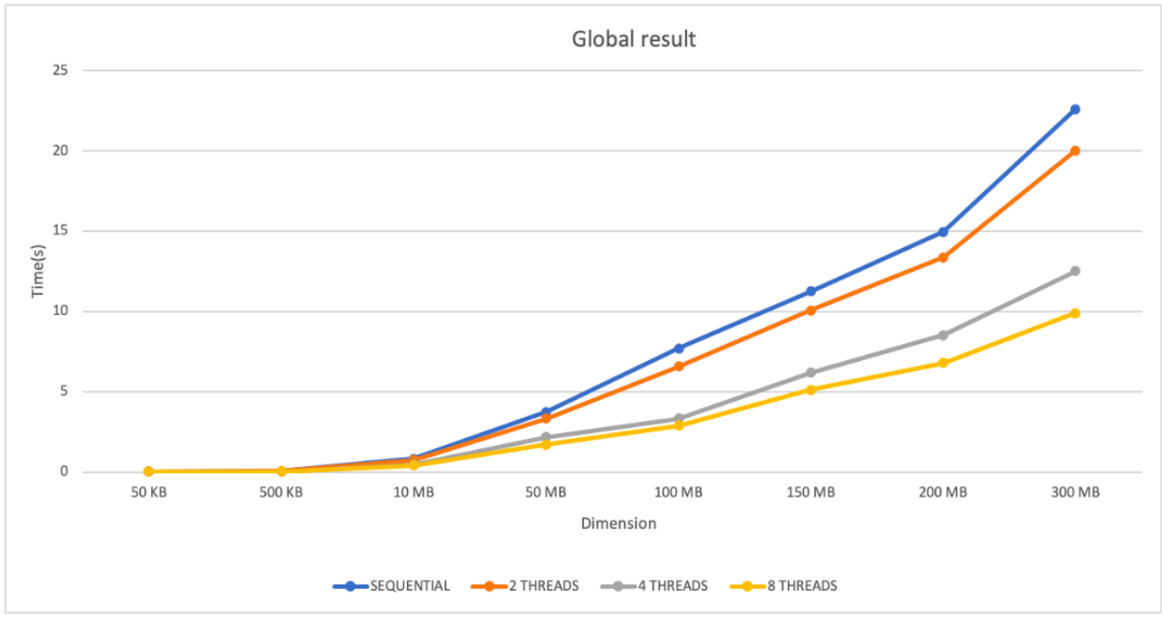
Parallel versions:

- The analysis was not focused by varying only text dimensions but also varying the number of threads used.
- 2, 4 and 8 threads were considered.
- Over the maximum number of threads chosen, computational times don't improve more significantly.
- For each parallel version speed up was evaluated.

Results

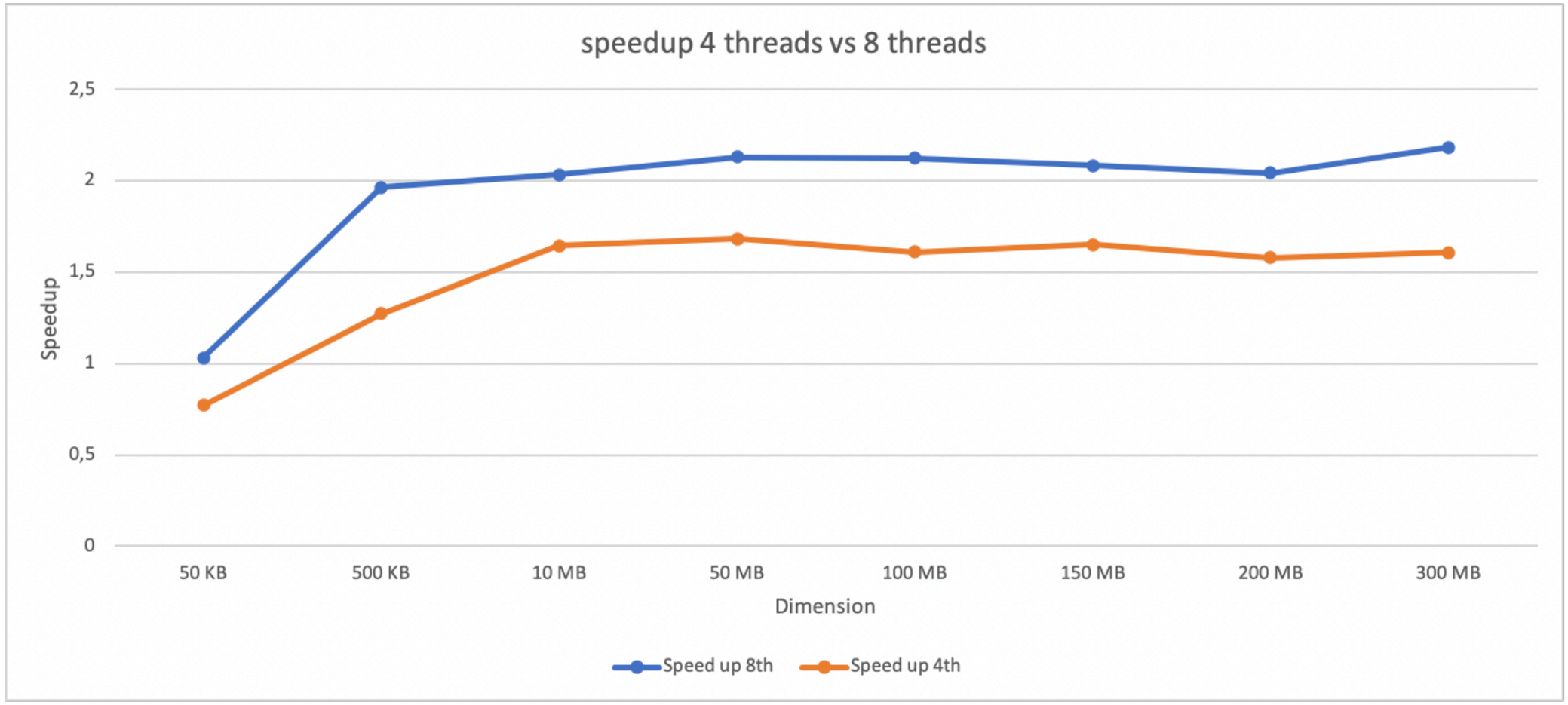


Bigrams



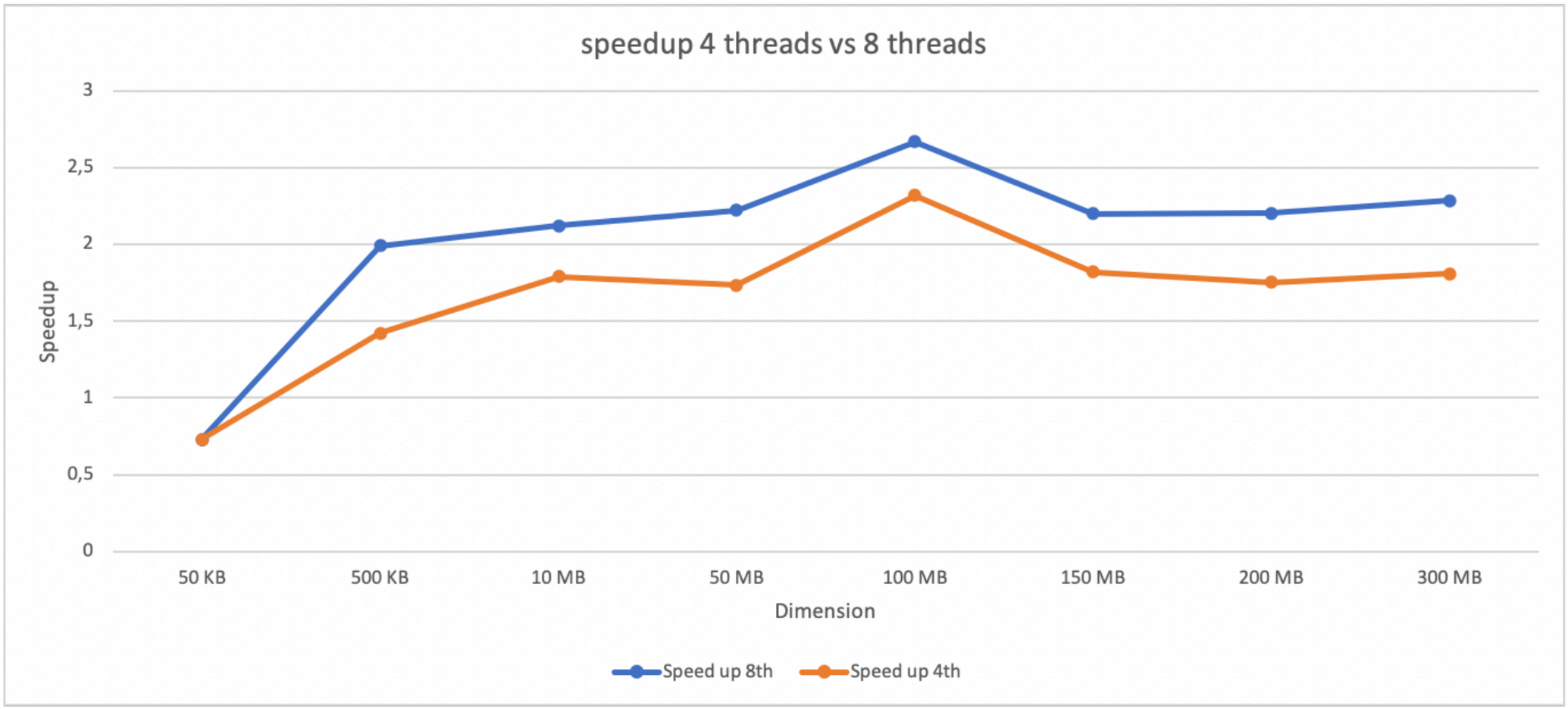
Trigrams

Results



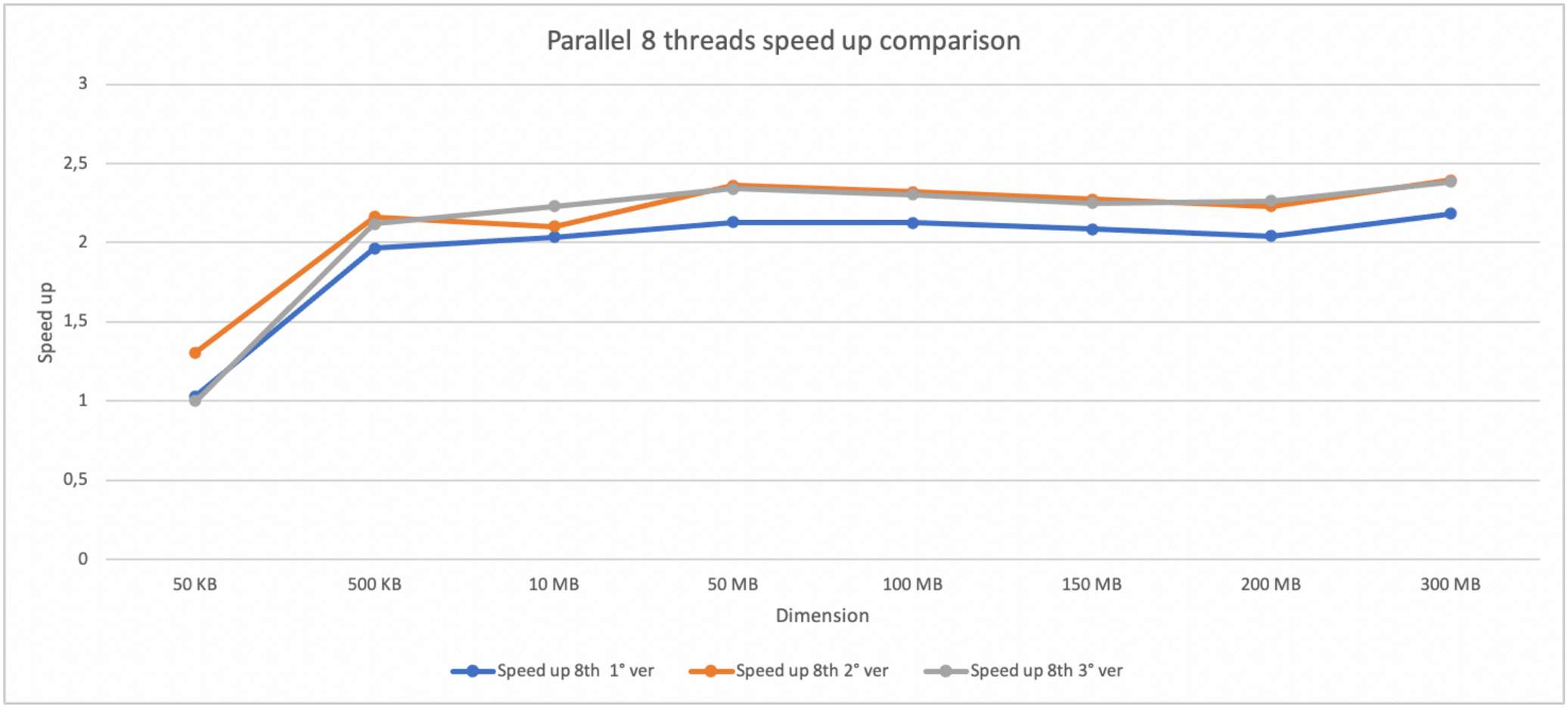
Speedup bigrams: parallel 4/8 threads

Results



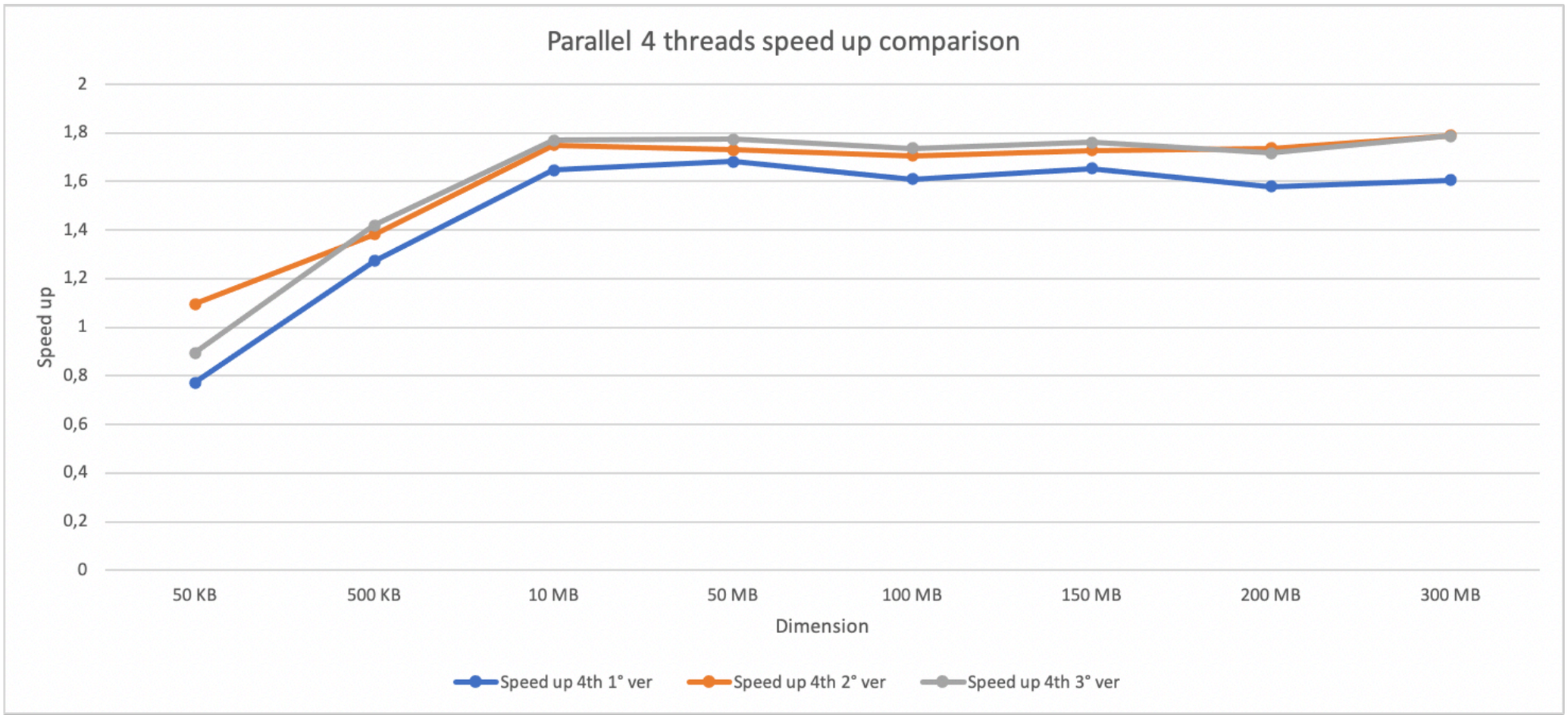
Speedup trigrams: parallel 4/8 threads

Results



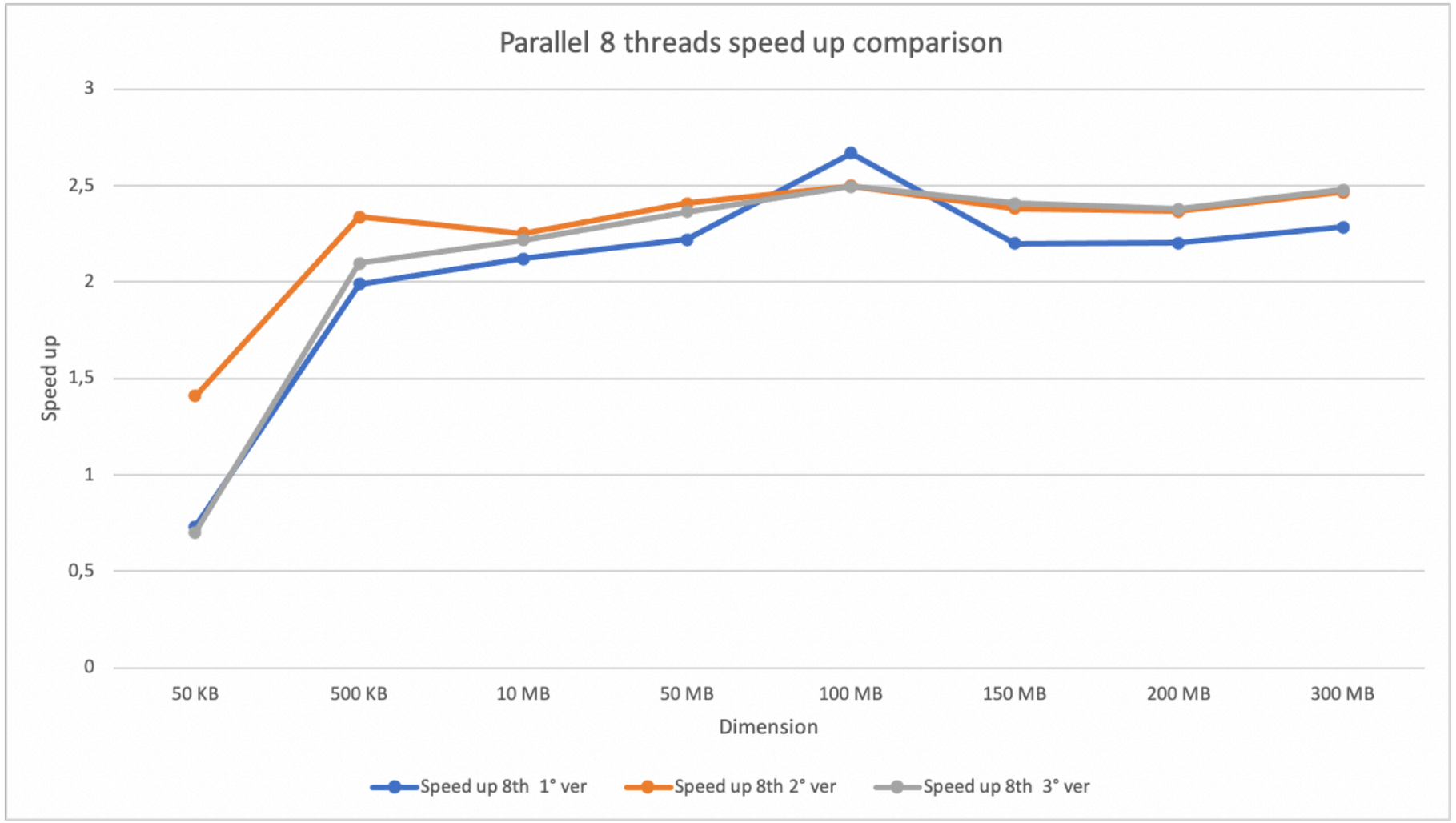
8 threads speedup bigrams comparison between the three different parallel approaches

Results



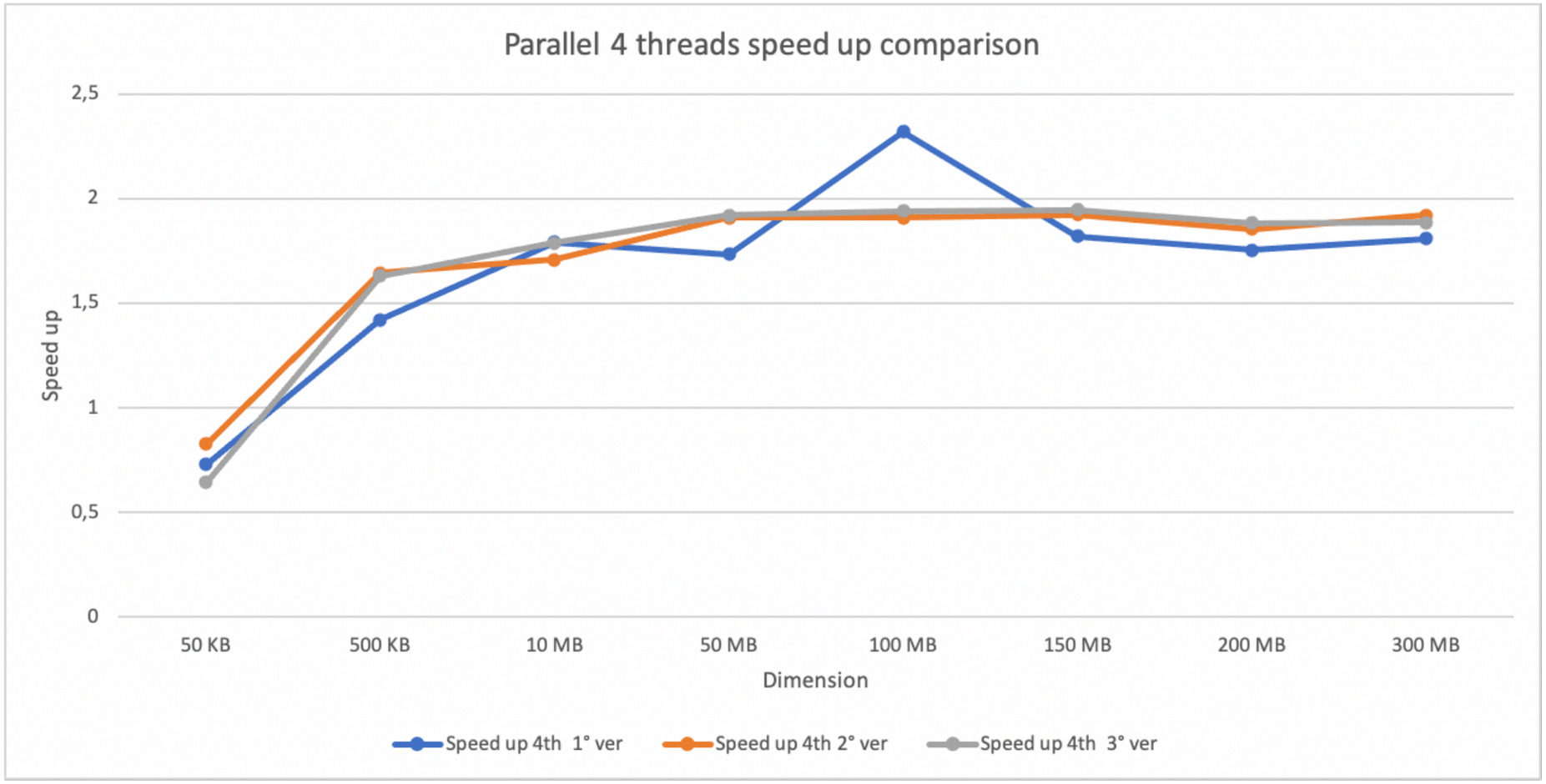
4 threads speedup bigrams comparison between the three different parallel approaches

Results



8 threads speedup trigrams comparison between the three different parallel approaches

Results



4 threads speedup trigrams comparison between the three different parallel approaches

UNIVERSITÀ
DEGLI STUDI
FIRENZE

Bigrams and trigrams

Andrea Simioni