MINHASH PROBLEM

INTRODUZIONE AL MINHASH

- MinHash è una tecnica algoritmica per stimare velocemente il grado di somiglianza tra due insiemi.
- Solitamente è applicato su larga scala in grandi problemi di raggruppamenti di documenti per somiglianza dei loro insiemi di parole.
- Inizialmente usato dal motore di ricerca AltaVista per individuare duplicati ed eliminarli dai risultati ricerca.
- Implementato da Andrei Broder nel 1997.

LA JACCARD SIMILARITY

Per il calcolo della somiglianza tra insiemi si usa comunemente il coefficiente di similarità di Jaccard, che è definito dal rapporto tra la dimensione dell'intersezione e la dimensione dell'unione degli insiemi campionari

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

OBIETTIVO MINHASH

- L’obiettivo del MinHash è di stimare velocemente J(*A*,*B*) senza calcolare esplicitamente l’intersezione e l’unione.
- Tramite una funzione di hash e delle permutazioni applicate a insiemi di dati, di cui vengono ricavati dei valori di hash minimi, il minHash produce una stima (probabilistica) approssimativa dell’Indice di Jaccard.

I TASK INDIVIDUATI

- “List dir” recupero da un path dato dei path dei files da confrontare.
- “tokenizer” recupero del contenuto di un file e pulizia del testo da caratteri spurii.
- “Shingle extract” creazione degli shingles dato un testo
- “Get Signatures”creazione delle signature tramite permutazioni di hash dati gli shingles.
- “Get Similarities” tramite un’opportuna comparazione vengono stampate e individuate le somiglianze.

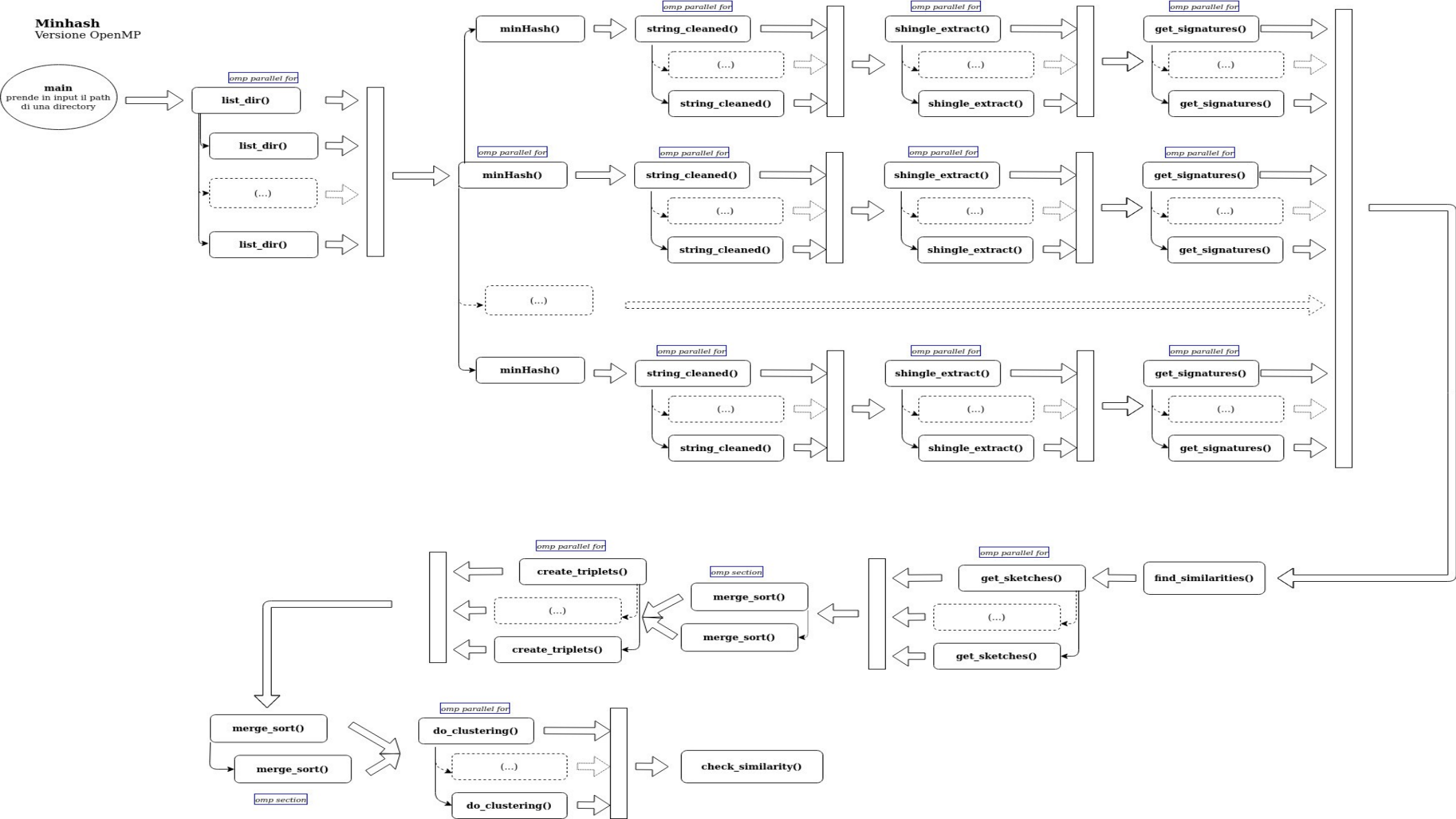
ALGORITMI PARALLELI UTILIZZATI

- OMP e parallelismo sui dati
- Pthreads e parallelismo sui dati
- Approccio con l’utilizzo di tutte e due le librerie assieme e parallelismo basato principalmente sui tasks.

OMP

- Propensione della libreria a parallelizzare algoritmi seriali già diviso in piccoli tasks.
- Parallelizzazione sui dati
- Utilizzo dell’istruzione parallel e ricerca delle critical section
- Utilizzo delle istruzione reduce e atomic per risolvere le problematiche dovute alla paralellizzazione.

Minhash
Versione OpenMP

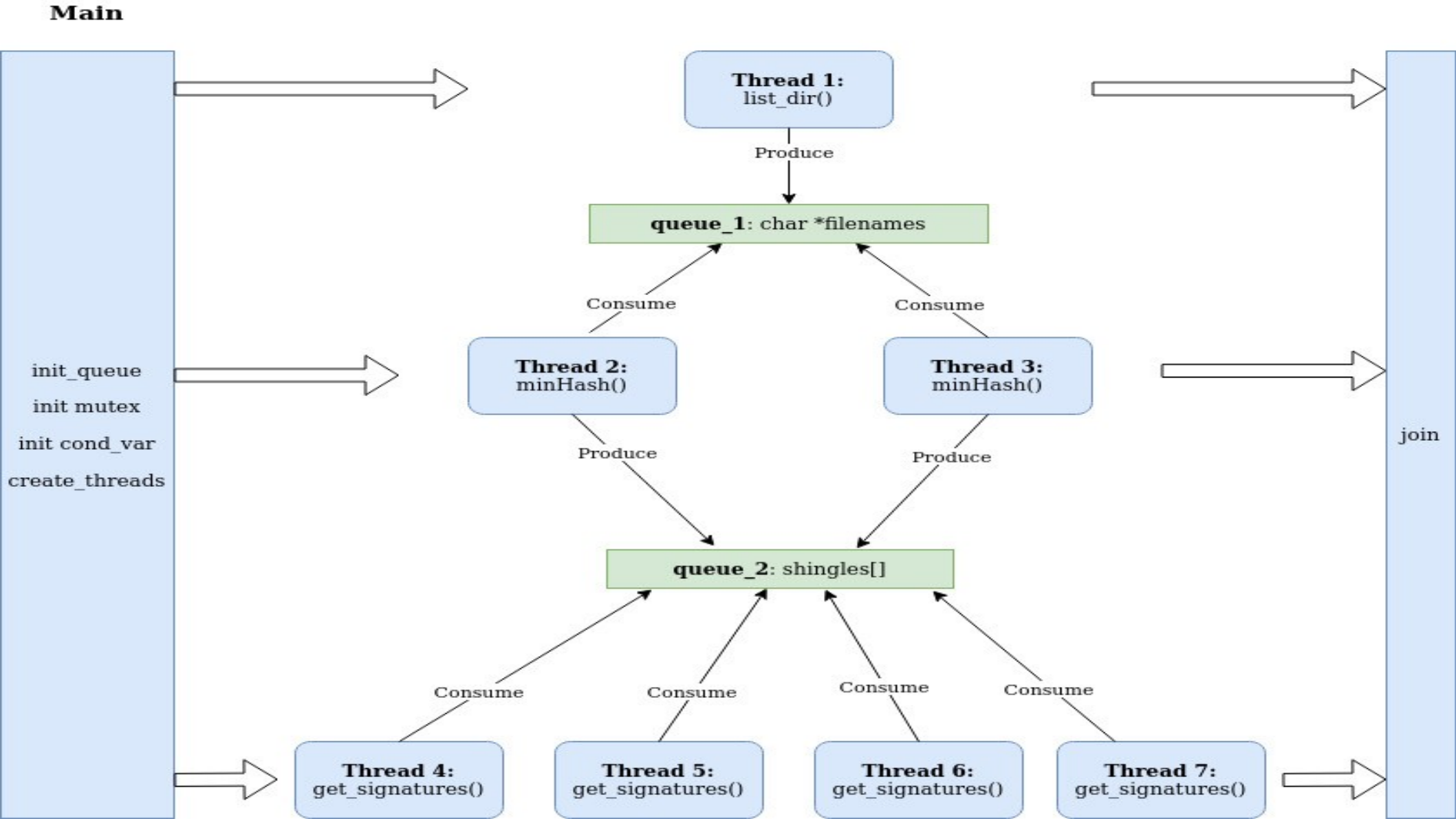


PTHREAD

- Parallelizzazione sui dati
- Approccio più a basso livello rispetto a OMP con uso d strutture dati apposite per comunicare con i thread creati.
- Individuazione delle barrier (pthread_join)
- Uso di mutex e individuazione delle sezione critiche

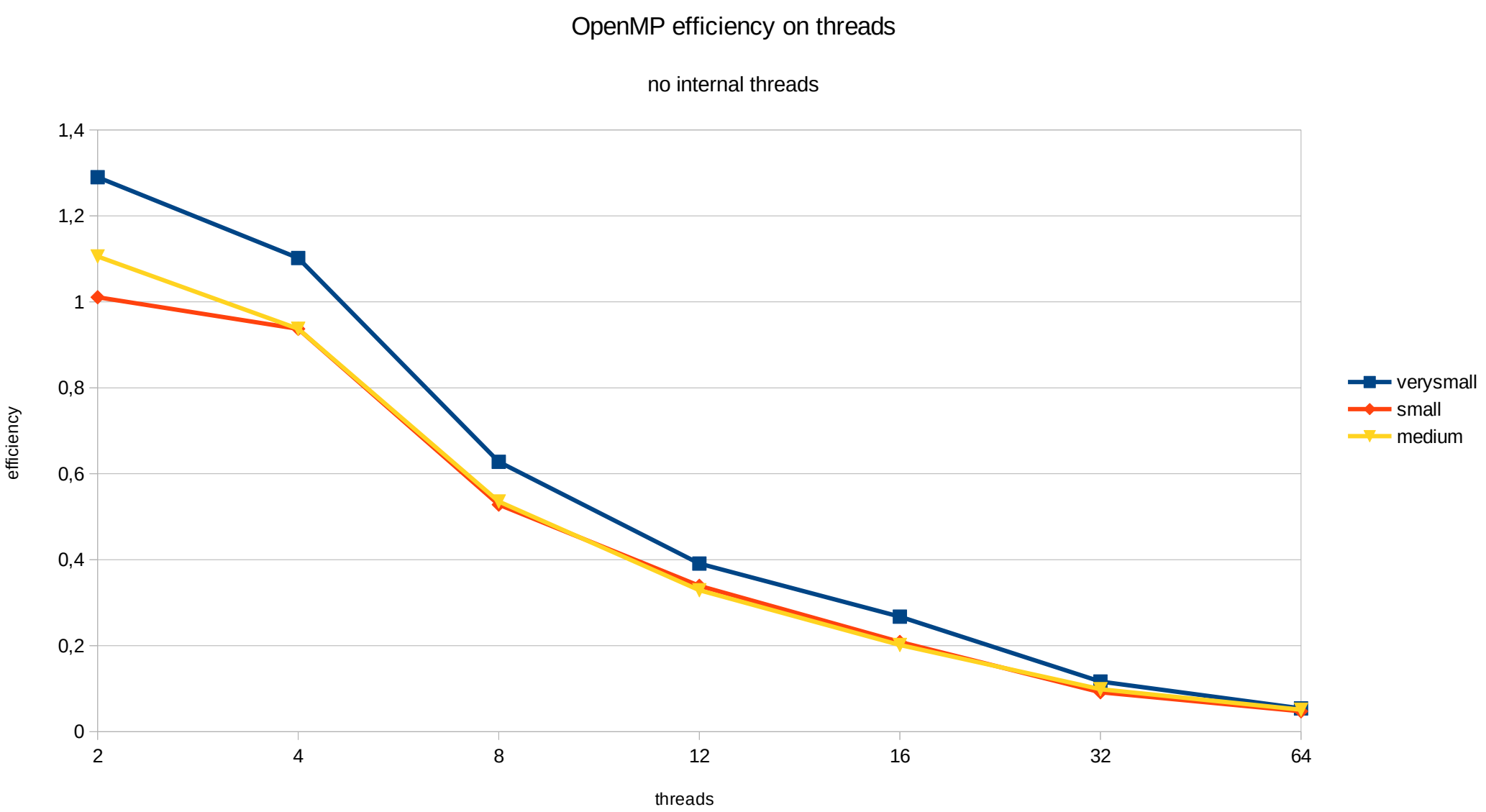
Produttore Consumatore

- In questa versione è stato usata principalmente la libreria pthread, solo nel task più costoso si è deciso di usare omp.
- Visione dei task come Produttori e Consumatori di strutture dati definite
- Parallelizzazione sui Task
- Struttura dati della coda per comunicare tra task


Main
Thread 1:
list_dir()
Produce
queue_1: char *filenames
Consume
Consume
Thread 2:
minHash()
Thread 3:
minHash()
init_queue
init mutex
init cond_var
create_threads
Produce
Produce
queue_2: shingles[]
Consume
Consume
Consume
Consume
Thread 4:
get_signatures()
Thread 5:
get_signatures()
Thread 6:
get_signatures()
Thread 7:
get_signatures()
join

TEST

- Raccolta dati basata sulle misurazioni dei wall time
- Controllo degli errori sulle signature create nella versione seriale
- Speed-up
- Efficienza e scalabilità


OpenMP efficiency on threads
no internal threads
1,4
1,2
1
0,8
0,6
0,4
0,2
0
efficiency
2
4
8
12
16
32
64
threads
verysmall
small
medium

OpenMP efficiency on threads

4 nested threads (Esempio 8 = 2 main threads * 4 nested threads)