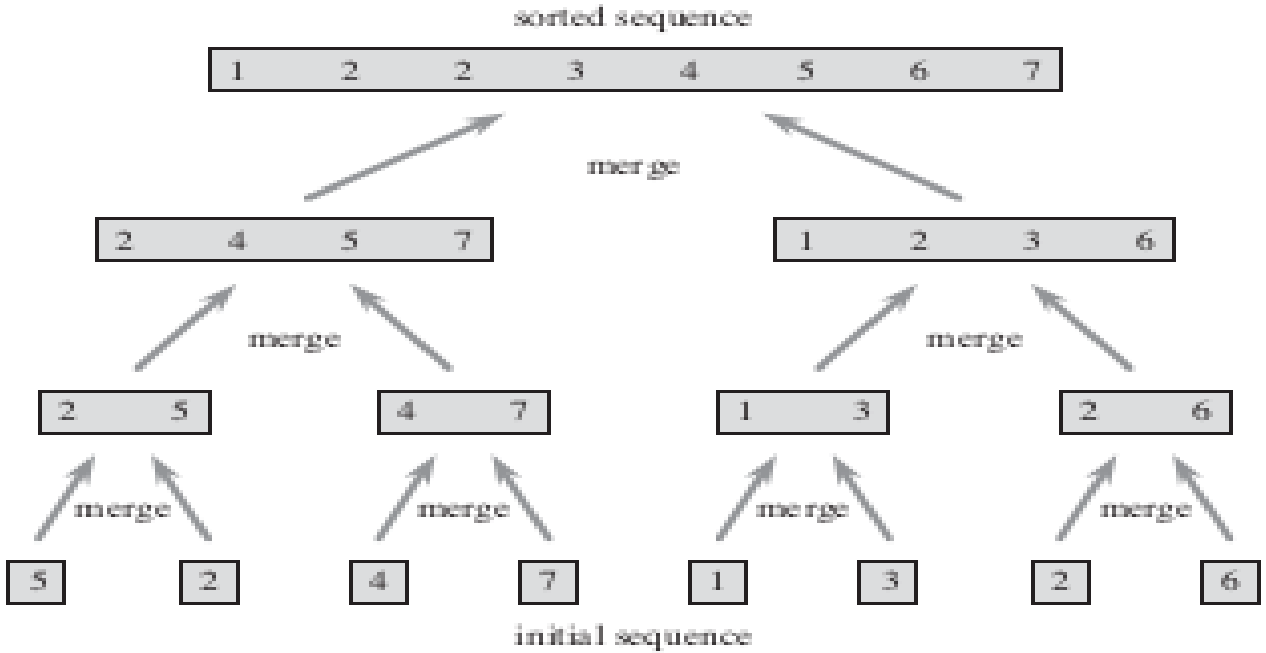
Confronto delle performance tra l’implementazione single-threaded e multi-threaded dell’algoritmo merge sort

Marco Carlo Feliciano, Università degli Studi di Napoli Federico II

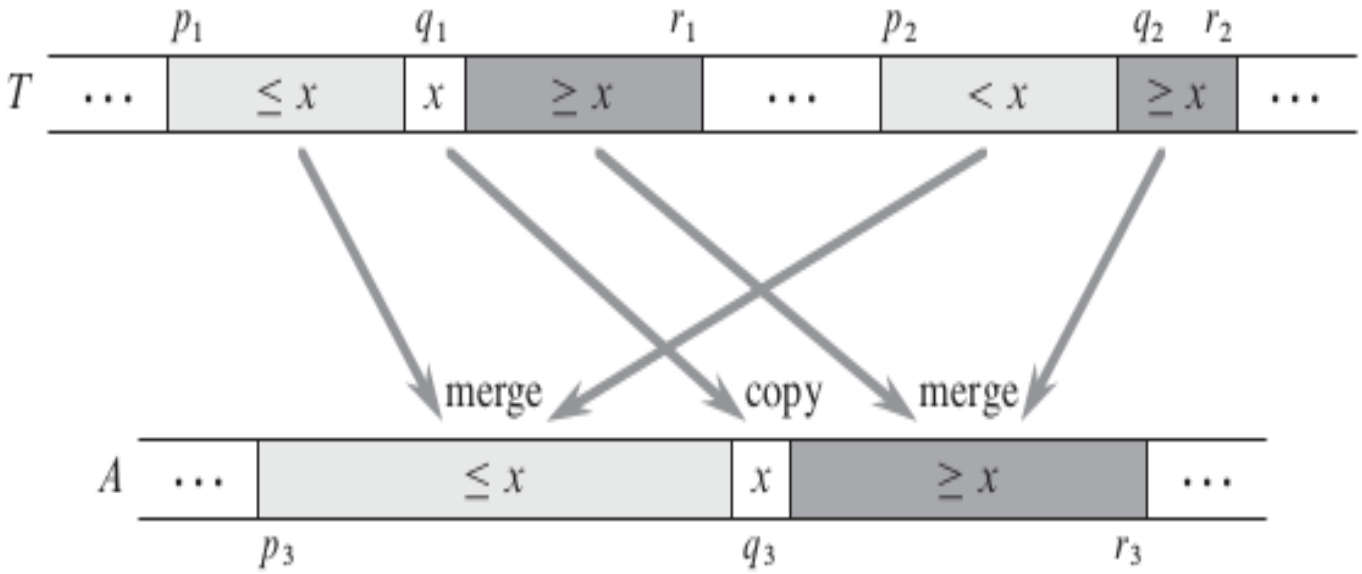
Versioni dell’algoritmo da confrontare

- Versione standard, con un solo thread
- Multi-threaded, con il merge seriale
- Multi-threaded, con il merge parallelo

Dall'implementazione standard a quella multi-threaded



Come parallelizzare la procedura merge



Documentazione relativa all’esecuzione dei test di performance

- Implementazione in Python
- CPU i5-6200U (2.3 GHz, 4 core)
- 8 GB di RAM
- Sistema operativo Fedora 33

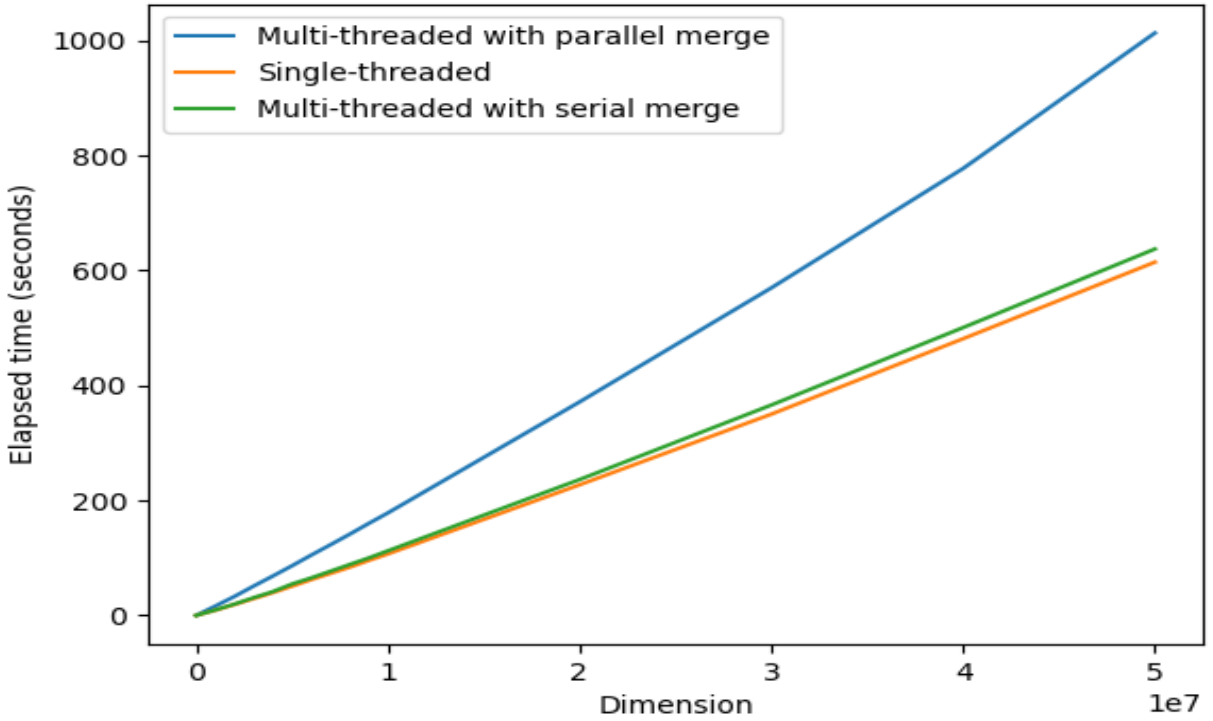
Test cases

Test case	Start	Step	End
Base	0	100	1100
Low	10000	10000	110000
Mid	1000000	1000000	10000000
High	10000000	10000000	60000000

Overview dei risultati

	Tempo di esecuzione (in secondi)		
Dimensione	Un solo thread	Multi-thread, merge seriale	Multi-thread, merge parallelo
1e3	0,0067	0,0091	0,0155
1e5	0,7692	0,7954	1,4511
1e6	8,8566	9,3121	16,0178
5*1e7	614	637	1013

Overview dei risultati



Interpretazione dei risultati

- I thread non sono schedulati in maniera indipendente, perché appartengono ad un unico processo
- Anche quando il processo è schedulato, il SO non gli dedica tutti i core
- Overhead dei thread “nascosto” nella notazione asintotica
- Al crescere della dimensione del problema, l’algoritmo di merge parallelo sarebbe più lento di quello seriale anche su un’architettura parallela dedicata
- Un singolo thread sfrutta meglio il principio di località dei dati e meccanismi di caching

Conclusione

La scelta tra single-threading e multi-threading va fatta con attenzione; casi d’uso di un algoritmo parallelo:

- Architettura sottostante che permette un parallelismo efficiente, come una GPU
- Problemi di dimensioni troppo grandi da gestire su un solo nodo, per i quali è necessario calcolo parallelo distribuito
- Applicazioni I/O bound, per avere un buon utilizzo complessivo delle risorse di sistema