- **La misura del tempo di calcolo è corretta?** tale misura è stata presa sulla computazione su stream con un solo worker e i moduli generatore e collettore collegati tramite i canali. Implementare la computazione sequenziale non in stream e verificare che il tempo di calcolo sia vicino a quello misurato precedentemente.
- **Il tempo di calcolo per una matrice 56x56 è troppo alto?**
-
- le misure sono prese male?
- è sbagliato il calcolo teorico del grado di parallelismo?
- la multicast non viene eseguita in parallelo? Il fatto che venga fatta la multicast nei workers, se non si sovrappongono le comunicazioni al calcolo, anche se sono di grana fine non servono a nulla. Il tempo di servizio ideale è 2 volte il tempo di comunicazione ma i worker non vanno in parallelo, quell'albero lì con queste macchine non fa nulla, è equivalente all'implementazione lineare.

1 Descrizione dell'applicazione

È stata realizzata una applicazione sul calcolo del prodotto matrice per vettore. La computazione considerata è su stream, gli elementi dello stream sono le matrici su cui effettuare il calcolo, mentre il vettore rimane costante per tutta l'esecuzione dell'applicazione. L'implementazione scelta fa uso del paradigma Data Parallel, in particolare la forma *Map*, in quanto ogni processo worker andrà ad eseguire un calcolo che è indipendente dall'attività degli altri processi. Si è scelto il partizionamento delle matrici per riga. Per realizzare le comunicazioni vengono usati i canali del supporto fornito, quindi si ha lo scambio dei puntatori alle strutture dati condivise. L'applicazione risulta seguire lo schema *multicast-compute-gather* in quanto per ogni matrice dello stream si eseguono le seguenti tre fasi:

a) la distribuzione del riferimento alla matrice corrente tra i processi worker, è compito di ogni worker effettuare il calcolo nella propria partizione della matrice,

b) il calcolo dei risultati parziali in ciascun worker,

c) la raccolta di tutti i risultati parziali dell'elemento.

La comunicazione collettiva *multicast* è implementata con una struttra ad albero binario mappato nell'insieme dei processi worker, le comunicazioni che costituiscono l'albero sono implementate per mezzo dei canali simmetrici offerti dal supporto; l'altra comunicazione collettiva, la *gather*, viene implementata per mezzo del canale asimmetrico in ingresso fornito dal supporto.

L'applicazione è fittizia, nel senso che non esistono dispositivi che generino e collezionino lo stream, per questo motivo l'applicazione è costituita, oltre che dai processi che realizzano la *map*, da altri due processi che, rispettivamente, generano e collezionano gli elementi dello stream. Tali due processi comunicano con il sottositema dei workers ($\Sigma 1^{(n)}$) per mezzo dei canali del supporto: il processo generatore è collegato tramite un canale simmetrico al worker radice dell'albero multicast, ogni processo worker è collegato al processo collettore per mezzo di un canale asimmetrico.

Riassumendo, ogni processo worker fa uso di 4 canali di comunicazione:

- tre sono simmetrici e trasportano gli elementi dello stream, uno dei quali è in ingresso dal processo padre nell'albero della multicast, gli altri due sono in uscita verso i processi radice dei due sottalberi della multicast,
- il quarto canale è asimmetrico in ingresso ed è usato in scrittura, per la comunicazione del risultato parziale al processo collezionatore.

È quindi possibile l'uso del supporto alle comunicazioni che fa uso della UDN in quanto, per ogni processo, il numero di canali non supera il numero di code hardware.

2 Analisi delle prestazioni

Di seguito viene descritta una breve analisi delle prestazioni attese dal benchmark. Dato che la frequenza dello stream è arbitraria, la stima del tempo di servizio del *map* permette di avere un primo dimensionamento del tempo di interarrivo, per diverse dimensioni dei dati, in modo tale da poter effettuare le prime misure dell'applicazione. Nella sezione successiva sono quindi proposte le misure del tempo di completamento dello stream e del tempo di servizio di $\Sigma 1^{(n)}$.

La macchina TILE*Pro*64 non dispone di processori di comunicazione, ne segue che le latenze di comunicazione dei canali sono pagate completamente nel tempo di servizio dei processi worker del sottosistema $\Sigma 1^{(n)}$. Si caratterizza perciò il tempo di servizio *ideale* ed *effettivo* del sottosistema *map* come segue:

$$T^{(n)}_{\Sigma 1\_\mathrm{id}} \;=\; T^{(n)}_S \;=\; T_{\mathrm{multicast}} + \frac{T_{\mathrm{cacl}}}{n} + T_{\mathrm{gather}} \;=\; 2 \cdot T_{\mathrm{sym\_send}} + \frac{T_{\mathrm{cacl}}}{n} + T_{\mathrm{asymin\_send}} \;=\; \Delta + \frac{T_{\mathrm{cacl}}}{n}$$

$$T^{(n)}_{\Sigma 1} \;= \max(\{T_A,\, T^{(n)}_S\})$$

Dove T_{calc} è il tempo medio impiegato per il calcolo della computazione sequenziale, ovvero il calcolo di una moltiplicazione matrice per vettore, e n è il grado di parallelismo dell'applicazione, inteso come il numero dei processi worker. Il rapporto tra T_{calc} e n esprime il tempo di servizio *ideale* di un processo worker scollegato dallo stream. Si è indicato con Δ la somma dei tempi spesi nelle comunicazioni da parte di un worker, tale latenza non è sovrapposta al tempo di calcolo nei processi worker.

Matrix Size	T_{calc} (μs)	T_{calc} (τ)
56x56	85.997340	74351.900000
168x128	848.096504	733250.424000
280x280	2360.060404	2040469.784000

Table 1: Tempi di calcolo della computazione sequenziale al variare della dimensione della matrice

Se è noto il valore del T_{calc} e di Δ allora per un generico tempo di interarrivo si ricava il grado di parallelismo ottimo, ovvero il minor grado di parallelismo che massimizza l'efficienza fornendo un fattore di utilizzazione unitario del sottosistema $\Sigma 1^{(n)}$:

$$n_{\text{opt}} \;=\; \min(\{n \in \mathbb{N} \mid T_S^{(n)} \leq T_A\}) \;=\; \left\lceil \frac{T_{\text{calc}}}{T_A - \Delta} \right\rceil$$

Dato che il tempo di interarrivo non è definito a priori ma è arbitrario, da tale formula è possibile ricavare il tempo di interarrivo uguale al tempo di servizio del *map* con il massimo grado di parallelismo esplicitabile dall'architettura.

Si pone quindi il problema di stimare il tempo di calcolo e il valore di Δ.

• Il T_{calc} viene stimato misurando il tempo medio impiegato per eseguire una moltiplicazione matrice per vettore in un singolo tile della macchina. I risultati di tale misura per dimensioni diverse della matrice sono mostrate in Tabella 1.

• Il valore di Δ può essere fornito in prima approssimazione dalle misure delle latenze di comunicazione effettuate con l'applicazione "ping-pong" per le due implementazioni dei canali. I risultati di tale misura sono riassunti nella Tabella 2.

Si osserva che le misure di tali parametri sono approssimazioni ottimistiche, è infatti prevedibile che durante l'esecuzione dell'applicazione *map* sia il tempo di calcolo dei worker che le latenze di comunicazione siano superiori ai valori stimati per mezzo delle applicazioni di misurazione, le quali usano un numero minimale di processi. Per la misura del T_{calc} viene eseguito un unico processo, per la misura delle latenze di comunicazione vengono eseguiti due processi che si scambiano messaggi usando il supporto fornito, in entrambi i casi le misure sono prese con un numero di conflitti minimo sia nelle reti di interconnessione sia alle memorie cache e ai controllori della memoria principale. Durante l'esecuzione della *map* i processi in gioco possono essere molti fino al massimo numero di processori utilizzabili nella macchina, ne segue un aumento dei conflitti alle reti e alle memorie rispetto a quelli che si verificano nei programmi di misurazione e ciò introduce overheads sia nel tempo di calcolo effettivo dei workers che nelle latenze di comunicazione.

Si calcola il tempo di servizio ideale con il grado di parallelismo massimo, $N = 59$, $T_{\text{S}}^{(\text{n})} = \frac{T_{\text{calc}}}{n} + \Delta$.

```
74352/59 + 181 = 1441  -> 4000
74352/59 + 481 = 1741  -> 7000
74352/59 + 725 = 1985

733250/59 + 181  = 12428 + 181 = 12609  -> 20000
                   12428 + 481 = 12909  -> 20000

2040470/59 + 181 = 34584 + 181 = 34765  -> 50000
```

Canale	*Lcom* (τ)
`ch_sym_udn`	55.722760
`ch_sym_sm_rdyack`	155.377520
`ch_asymin_udn`	69.509710
`ch_asymin_sm`	170.285120
`ch_asymin_sm_all`	414.157690

Canali usati	$T_\Delta(\tau)$
UDN only	180.95523
SM only	481.04016
SM only with all	724.91273

Table 2: Misure delle latenze dei canali di comunicazione rilevate con l'applicazione "ping-pong", nella quale i due processi sono eseguiti in due tile con distanza massima nella mesh

3 Misure sul benchmark

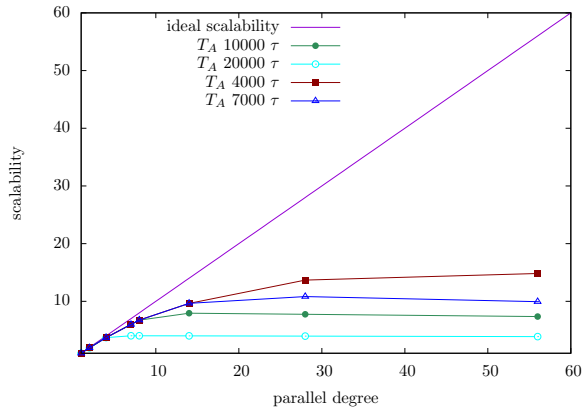
3.1 Variazione del tempo di interarrivo

Di seguito vengono proposti i grafici della scalabilità e del tempo di servizio del benchmark al variare del tempo di interarrivo. I parametri del benchmark che rimangono fissati sono l'implementazione dei canali usata, la dimensione delle matrici e la lunghezza dello stream. Viene proposto due diversi usi delle implementazioni: il solo utilizzo del supporto UDN e il solo utilizzo del supporto Memoria Condivisa.

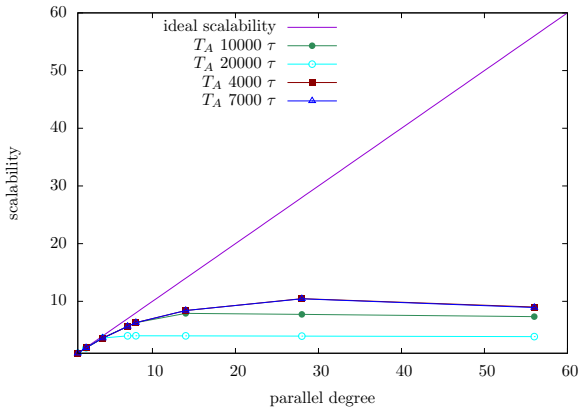
Figure 1: Grafici di scalabilità al variare del tempo di interarrivo

(a) Implementazione con solo UDN

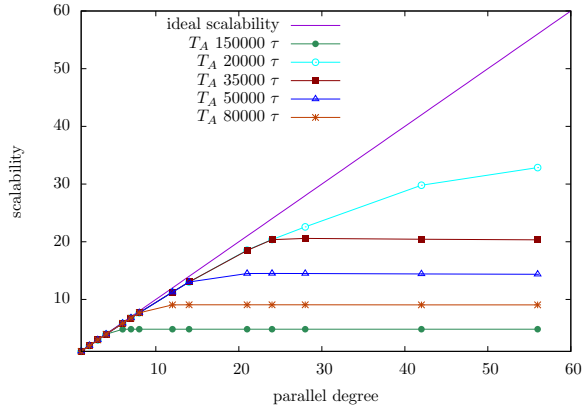
(b) Implementazione con solo SM



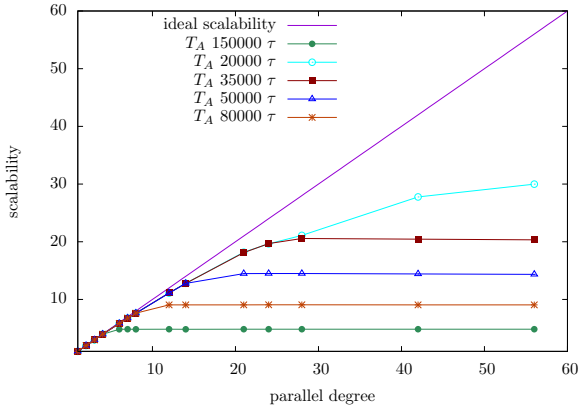
(a1) Scalabilita dell implementazione UDN con M=56 al variare del tempo di interarrivo



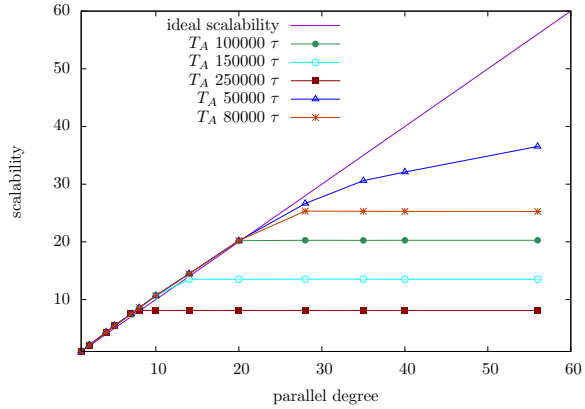
(b1) Scalabilita dell implementazione SM con M=56 al variare del tempo di interarrivo



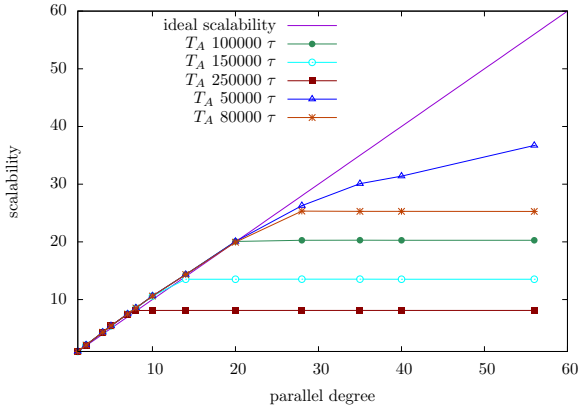
(a2) Scalabilita dell implementazione UDN con M=168 al variare del tempo di interarrivo



(b2) Scalabilita dell implementazione SM con M=168 al variare del tempo di interarrivo



(a3) Scalabilita dell implementazione UDN con M=280 al variare del tempo di interarrivo

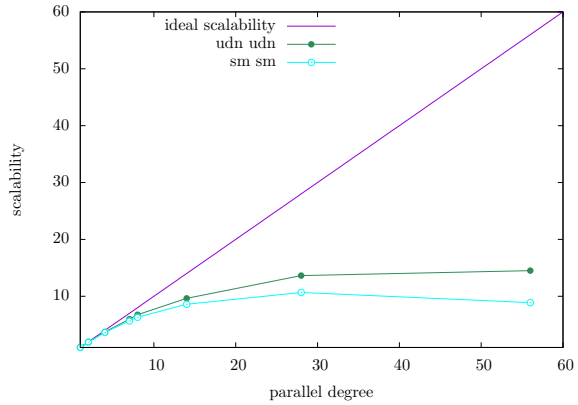


(b3) Scalabilita dell implementazione SM con M=280 al variare del tempo di interarrivo

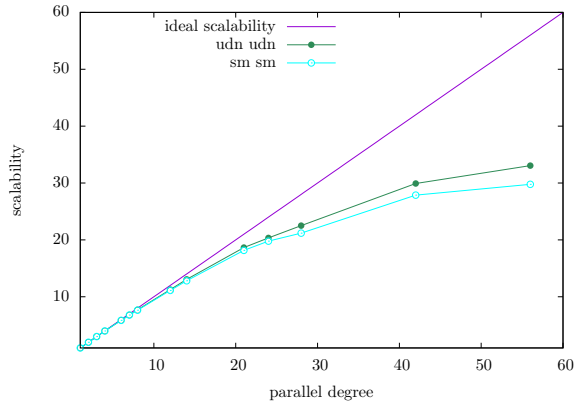
4 Confronto tra le due soulzioni con $\Sigma 1^{(n)}$ collo di bottiglia

Di seguito viene proposto il confronto della scalabilità del tempo di completamento del benchmark con l'uso delle due diverse soluzioni, il tempo di interarrivo è tale per cui il sottosistema $\Sigma 1^{(n)}$è collo di

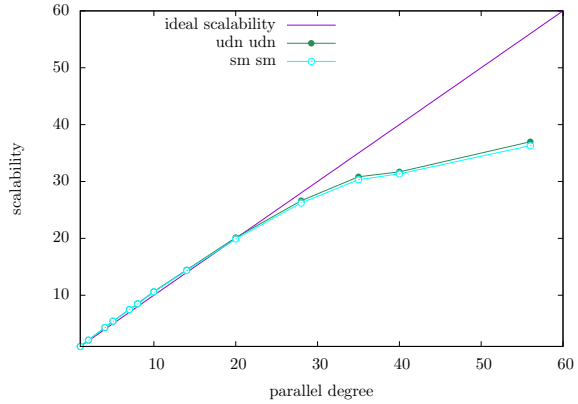
bottiglia.



(a) Confronto della scalabilita nelle diverse implementazioni, Ta=181, M=56



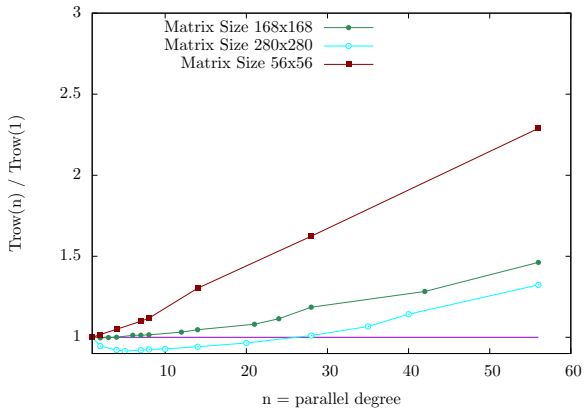
(b) Confronto della scalabilita nelle diverse implementazioni, Ta=181, M=168



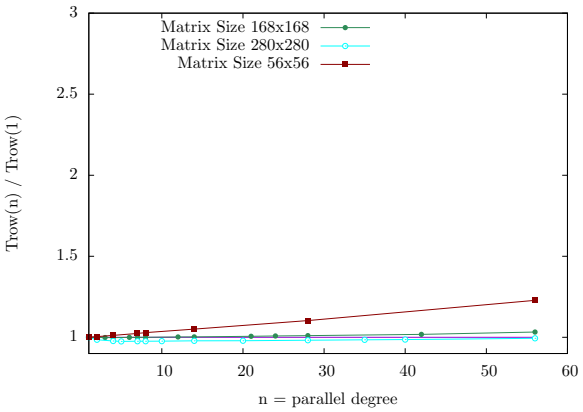
(c) Confronto della scalabilita nelle diverse implementazioni, Ta=181, M=280

5

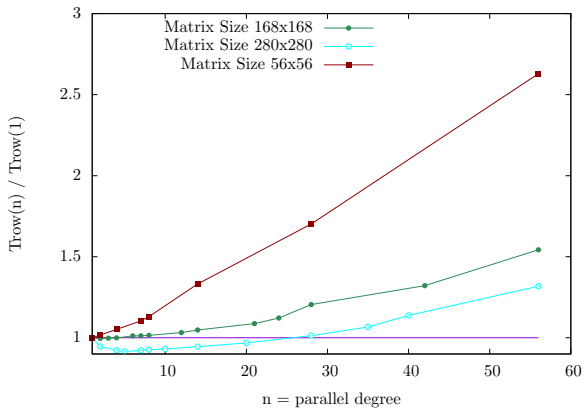
```
1  #define DONT_MOVE(EXP) \
2    atomic_compiler_barrier(); \
3    EXP; \
4    atomic_compiler_barrier();
5
6  generator_task(...) {
7    ...
8    pthread_barrier_wait(...);
9    DONT_MOVE(start_Tc = get_clock_cycle());
10   for (i=0; i<m; i++) {
11     a = get_clock_cycle();
12     sym_send(ch_out);
13     while (get_clock_cycle() - a < Ta)
14       ;
15   }
16   ...
17 }
18
19 worker_task(...) {
20   ...
21   pthread_barrier_wait(...);
22   for (i=0; i<m; i++) {
23     A = (int *)sym_receive(ch_in);
24     DONT_MOVE(
25       start_servicet = stop_servicet;
26       stop_servicet = get_clock_cycle();
27       sum_servicet = stop_servicet - start_servicet;
28     );
29     DONT_MOVE(start_multicast = get_clock_cycle());
30     if (ch_left != NULL) sym_send(ch_left, A);
31     if (ch_right != NULL) sym_send(ch_right, A);
32     DONT_MOVE(stop_multicast = get_clock_cycle(); ...);
33     for (j=0; j<g; j++) {
34       C[j*rank] = 0;
35       for (k=0; k<M; k++)
36         C[j*rank] = *(A+j*M+k) * B[k] + C[j*rank];
37     }
38     asymin_send(ch_out, C);
39   }
40   ...
41 }
42
43 collector_task(...) {
44   ...
45   pthread_barrier_wait(...);
46   for (i=0; i<m; i++) {
47     (void)asymin_receive(ch_in);
48   }
49   DONT_MOVE(stop_Tc = get_clock_cycle());
50   ...
51 }
```



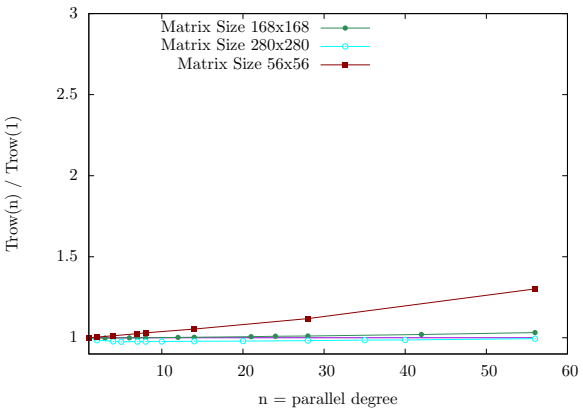
(d) Tempi di calcolo di una singola Row · Col con Ta=181, canali UDN e dati Int



(e) Tempi di calcolo di una singola Row · Col con Ta=181, canali UDN e dati Float



(f) Tempi di calcolo di una singola Row · Col con Ta=181, canali SM e dati Int

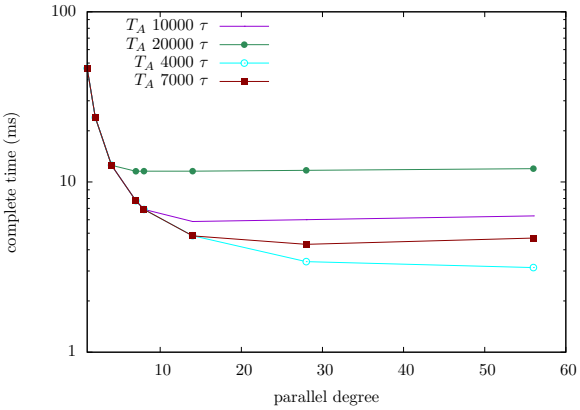


(g) Tempi di calcolo di una singola Row · Col con Ta=181, canali SM e dati Float

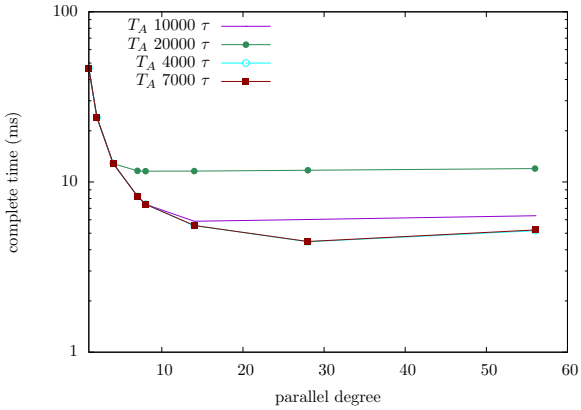
Figure 2: Grafici del tempo di completamento al variare del tempo di interarrivo

(h) Implementazione con solo UDN

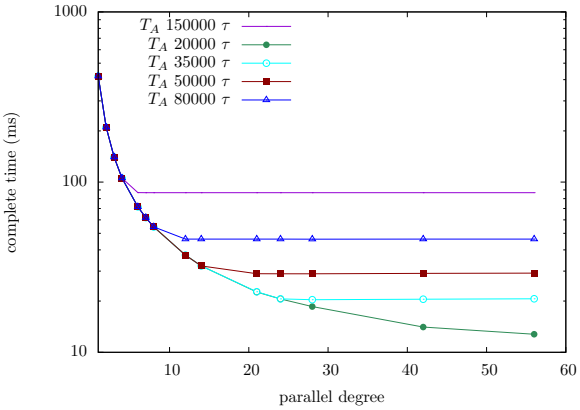
(i) Implementazione con solo SM



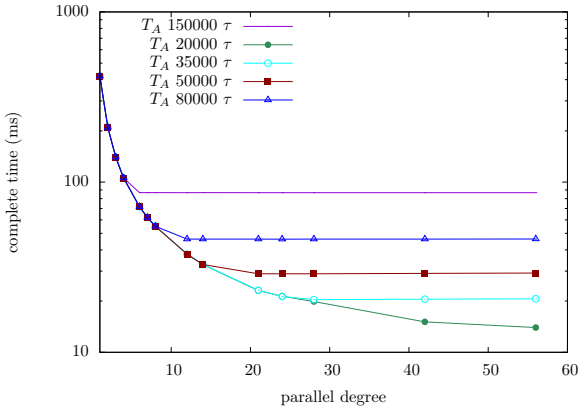
(h1) Tempo di completamento dell implementazione UDN con M=56 al variare del tempo di interarrivo



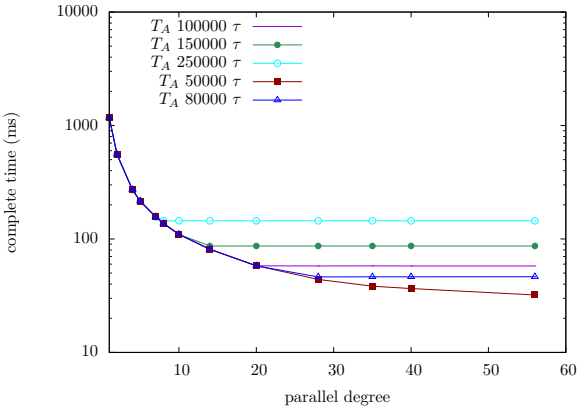
(i1) Tempo di completamento dell implementazione SM con M=56 al variare del tempo di interarrivo



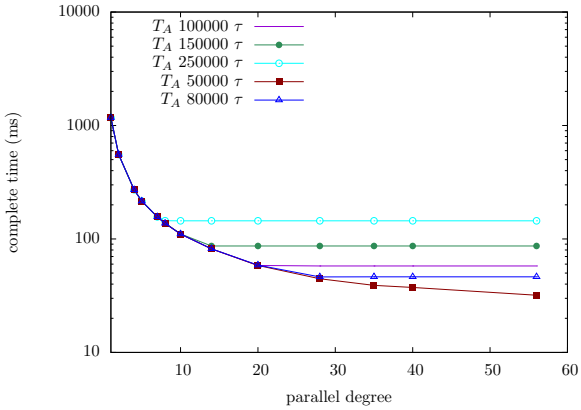
(h2) Tempo di completamento dell implementazione UDN con M=168 al variare del tempo di interarrivo



(i2) Tempo di completamento dell implementazione SM con M=168 al variare del tempo di interarrivo



(h3) Tempo di completamento dell implementazione UDN con M=280 al variare del tempo di interarrivo

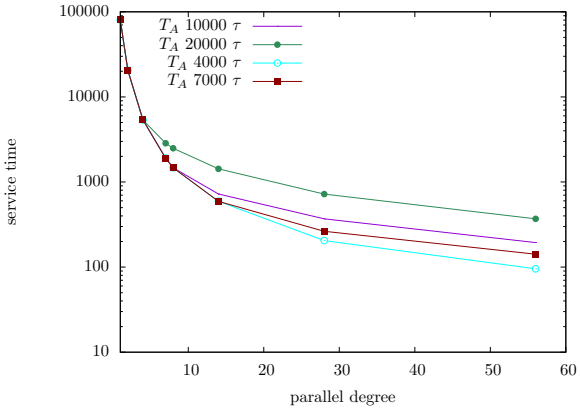


(i3) Tempo di completamento dell implementazione SM con M=280 al variare del tempo di interarrivo

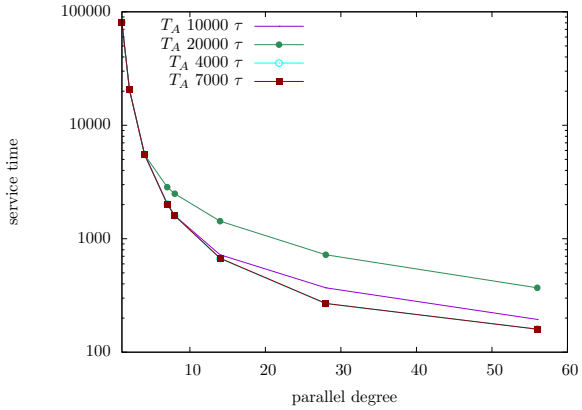
Figure 3: Grafici del tempo di servizio al variare del tempo di interarrivo

(a) Implementazione con solo UDN

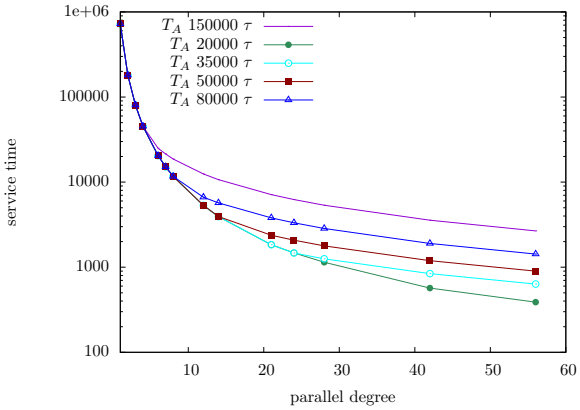
(b) Implementazione con solo SM



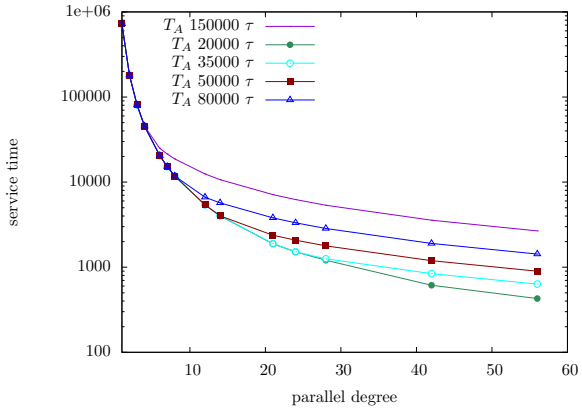
(a1) Tempo di servizio dell implementazione UDN con M=56 al variare del tempo di interarrivo



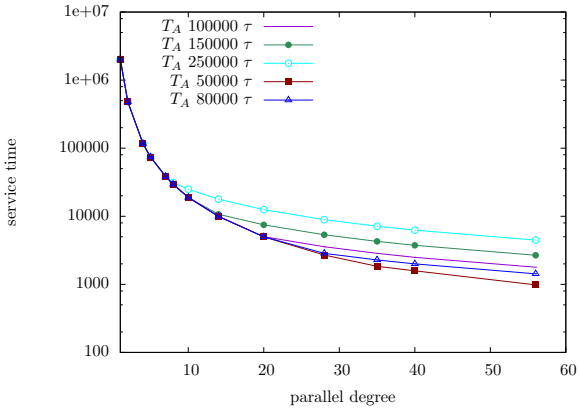
(b1) Tempo di servizio dell implementazione SM con M=56 al variare del tempo di interarrivo



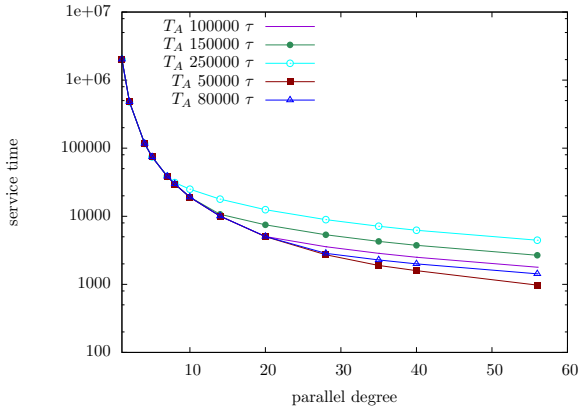
(a2) Tempo di servizio dell implementazione UDN con M=168 al variare del tempo di interarrivo



(b2) Tempo di servizio dell implementazione SM con M=168 al variare del tempo di interarrivo



(a3) Tempo di servizio dell implementazione UDN con M=280 al variare del tempo di interarrivo

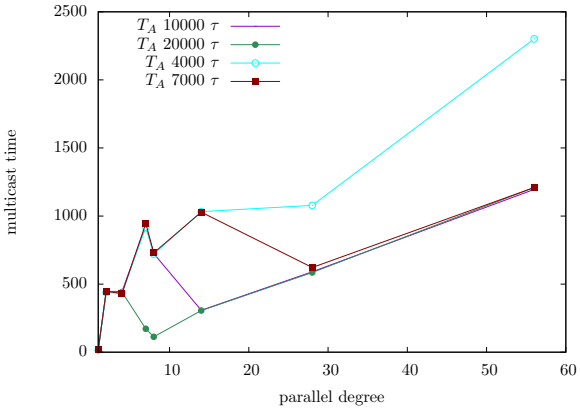


(b3) Tempo di servizio dell implementazione SM con M=280 al variare del tempo di interarrivo

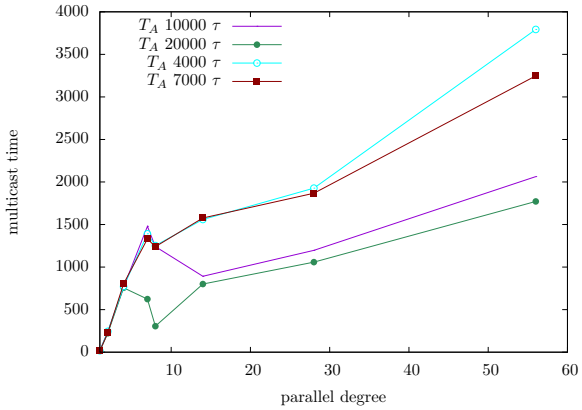
Figure 4: Grafici del tempo di multicast al variare del tempo di interarrivo

(a) Implementazione con solo UDN

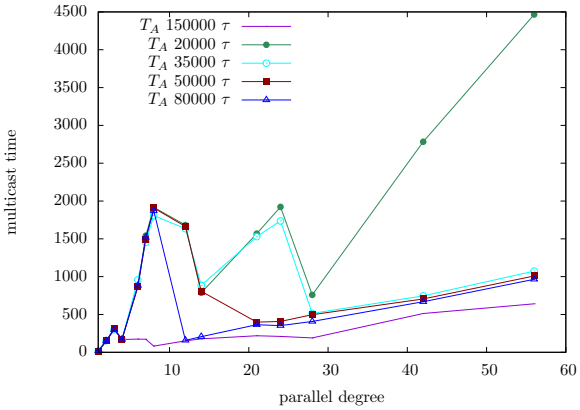
(b) Implementazione con solo SM



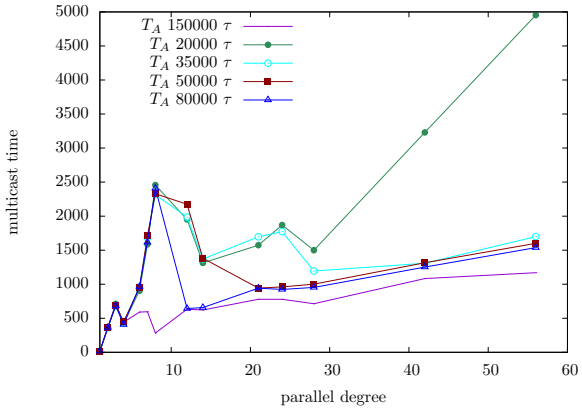
(a1) tempo di multicast dell implementazione UDN con M=56 al variare del tempo di interarrivo



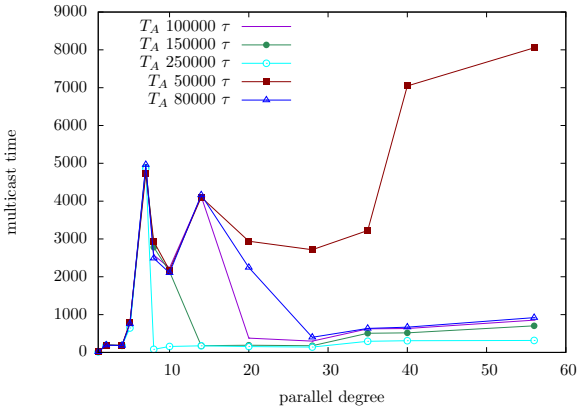
(b1) tempo di multicast dell implementazione SM con M=56 al variare del tempo di interarrivo



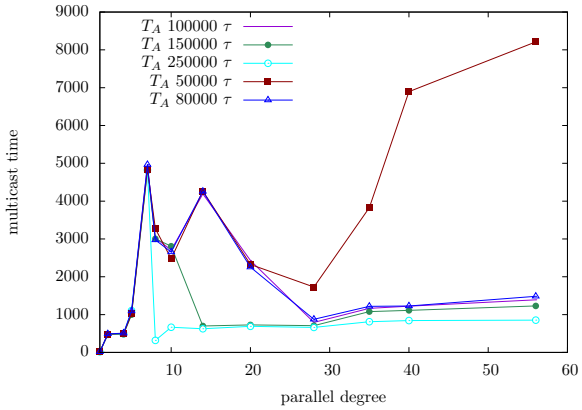
(a2) tempo di multicast dell implementazione UDN con M=168 al variare del tempo di interarrivo



(b2) tempo di multicast dell implementazione SM con M=168 al variare del tempo di interarrivo



(a3) tempo di multicast dell implementazione UDN con M=280 al variare del tempo di interarrivo



(b3) tempo di multicast dell implementazione SM con M=280 al variare del tempo di interarrivo