1

Alessandro Serra

December 2021

1 Ring

Il programma consiste in un anello costituito da un numero di elementi pari al numero di processi coinvolti nell'esecuzione. Ad ogni iterazione il processo con rank pari all'iterazione corrente manda un messaggio a sinistra e uno a destra con un suo specifico tag e i processi si scambiano i 2 messaggi fino a quando non tornano al processo che li ha inviati. Dopo un esecuzione del programma dunque il numero di messaggi scambiati sarà $2 \cdot N^2$ con N il numero di processi coinvolti. Il grafico seguente mostra l'andamento del tempo al variare del numero di processi: Andando poi a modellare la bandwidth si ottiene il seguente grafico.

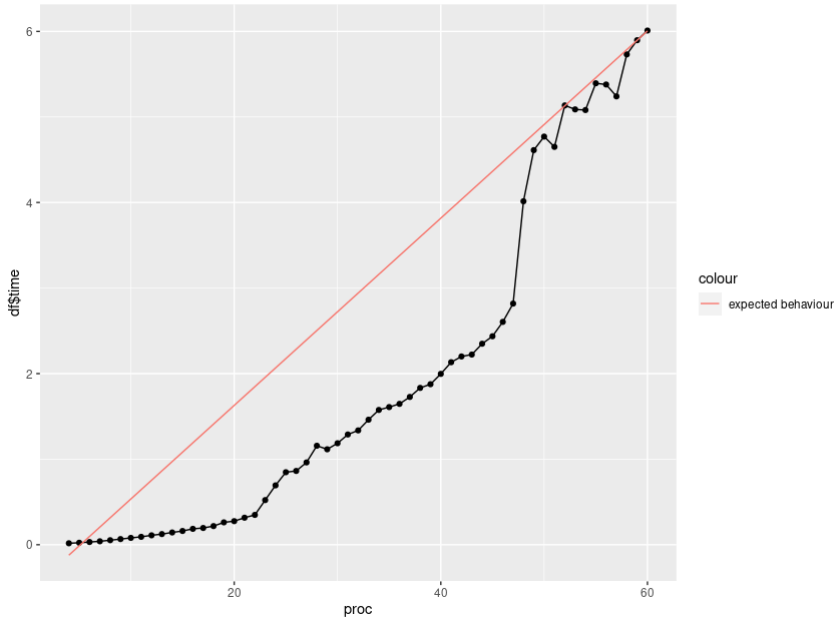

Figure 1: tempo su numero processori

Nel nostro modello all'aumento del numero di processi corrisponde un aumento lineare dei Bytes scambiati da ogni processo (ognuno scambia $2 \cdot N$ messaggi), per tanto non siamo in un contesto né di weak scaling né di strong scaling, è possinile tuttavia notare un crollo delle perfomance a partire dall'esecuzione del programma con 26 processi e poi nuovamente con 48 processi. Il numero di processi per il quale si verifica questo fenomeno ci suggerisce che

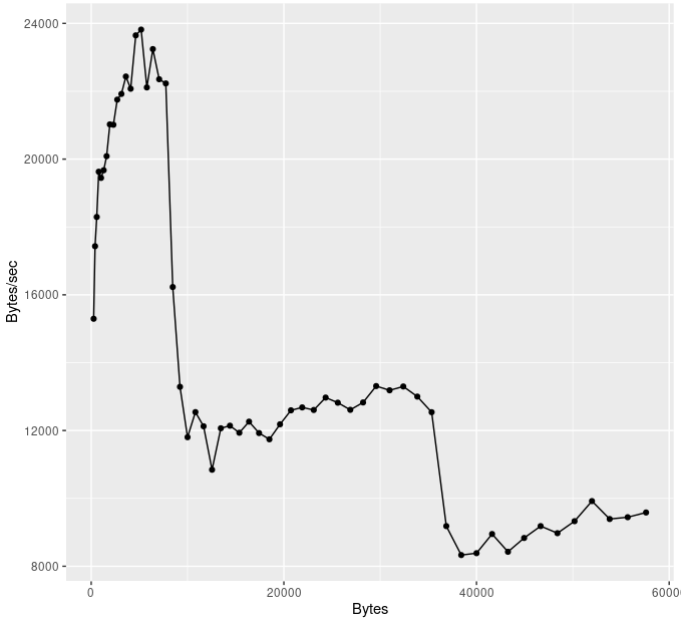

Figure 2: Banda su bytes scambiati

potrebbe essere dovuto ad una variazione della latenza poichè almeno due dei processi coinvolti si trovano rispettivamente in socket diversi e nodi diversi. Non è possibile approssimare il comportamento delle performance con un modello lineare, più nel dettaglio:

	n	time	Mb	Mb/s	Penalty
1	4	0.0167382	256	15294.357	1.000000
2	5	0.0229452	400	17432.840	1.096663
3	6	0.0314850	576	18294.426	1.254018
4	7	0.0399421	784	19628.412	1.363591
5	8	0.0526583	1024	19446.127	1.572998
6	9	0.0658818	1296	19671.594	1.749340
7	10	0.0796568	1600	20086.170	1.903593
8	11	0.0920879	1936	21023.392	2.000604
9	12	0.1096500	2304	21012.312	2.183628
10	13	0.1243060	2704	21752.771	2.285072
11	14	0.1430550	3136	21921.639	2.441891
12	15	0.1604690	3600	22434.240	2.556531
13	16	0.1855580	4096	22073.961	2.771475
14	17	0.1955410	4624	23647.215	2.748781
15	18	0.2176560	5184	23817.400	2.889678
16	19	0.2612090	5776	22112.561	3.285381
17	20	0.2753570	6400	23242.554	3.290163
18	21	0.3156470	7056	22354.085	3.591978
19	22	0.3483510	7744	22230.451	3.783952

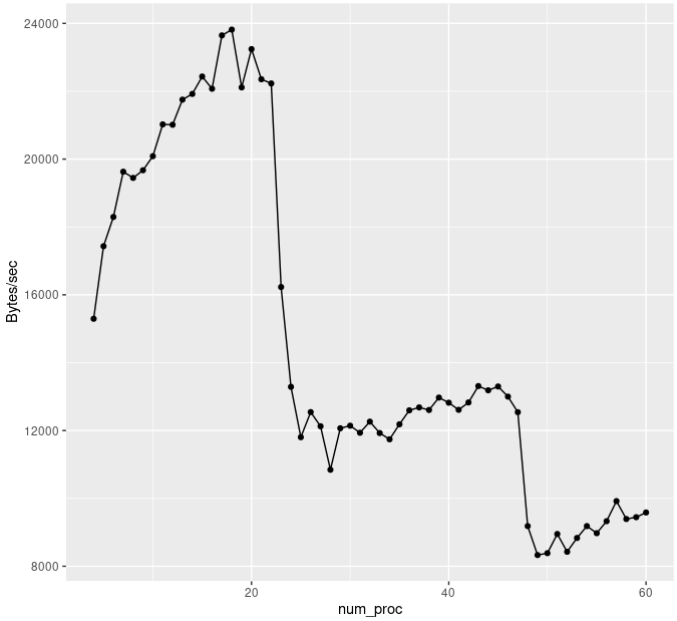

Figure 3: Banda su processi coinvolti

20	23	0.5215250	8464	16229.327	5.418743
21	24	0.6935790	9216	13287.600	6.906149
22	25	0.8472450	10000	11802.961	8.098792
23	26	0.8623250	10816	12542.835	7.925905
24	27	0.9619480	11664	12125.396	8.514106
25	28	1.1567600	12544	10844.082	9.872712
26	29	1.1152100	13456	12065.889	9.189881
27	30	1.1859400	14400	12142.267	9.446974
28	31	1.2884000	15376	11934.182	9.932081
29	32	1.3362100	16384	12261.546	9.978746
30	33	1.4611300	17424	11925.017	10.580986
31	34	1.5753000	18496	11741.256	11.072243
32	35	1.6087400	19600	12183.448	10.984216
33	36	1.6463200	20736	12595.364	10.928561
34	37	1.7272800	21904	12681.210	11.156096
35	38	1.8328700	23104	12605.368	11.526549
36	39	1.8761800	24336	12971.037	11.496381
37	40	1.9969000	25600	12819.871	11.930196
38	41	2.1329500	26896	12609.766	12.432202
39	42	2.2002700	28224	12827.517	12.519239
40	43	2.2226700	29584	13310.118	12.352583
41	44	2.3492700	30976	13185.372	12.759436
42	45	2.4364500	32400	13298.036	12.938866
43	46	2.6044200	33856	12999.439	13.530207
44	47	2.8184500	35344	12540.226	14.330578

```
45 48 4.0144100 36864 9182.919 19.986269
46 49 4.6121200 38416 8329.358 22.493434
47 50 4.7704000 40000 8385.041 22.800062
48 51 4.6500000 41616 8949.677 21.788835
49 52 5.1348100 43264 8425.628 23.597841
50 53 5.0881800 44944 8833.021 22.942347
51 54 5.0806900 46656 9183.005 22.484342
52 55 5.3935000 48400 8973.765 23.434691
53 56 5.3791500 50176 9327.868 22.954977
54 57 5.2403300 51984 9919.986 21.970251
55 58 5.7310900 53824 9391.582 23.613506
56 59 5.8969100 55696 9444.947 23.884916
57 60 6.0099700 57600 9584.074 23.937142
```

Nella quarta colonna si è calcolato $\frac{P_4 \cdot \frac{N}{4}}{P_N}$ dove N rappresenta il numero di processi coinvolti e P_N la performance ottenuta con N processi. In altre parole *penalty* rappresenta quanto il comportamento osservato si discosta da un porblema perfettamente scalabile e come si evince dai le performance sono fortemente non lineari.

2 Somma di Matrici

Il programma su cui si è effettuata l'analisi genera due matrici e le divide in subdomains che vengono inviati ad ogni core. Ogni processo effettua le somma delle due sottomatrici ricevute e restituisce il risultato al rank 0 che ricombina poi i dati ottenuti nella matrice di partenza. Il programma è in grado di disitribuire lungo ogni asse un numero di processi che sia divisore della dimensione della matrice lungo quell'asse per tanto sono state omesse le distribuzioni di processori per cui ciò non era possibile. In generale il variare della topologia dei processori influisce sulle performance di quei programmi che implementano una divisione in subdomains del problema iniziale e scambiano fra di loro dati. Questo è dovuto in gran parte al fatto che a seconda che la divisione in subdomain sia in 1D, 2D o 3D si avrà una variazione del workload dovuto allo scambio di dati, e dunque il communication overhead impatterà negativamente sulle performance. Si può dimostrare inoltre che:

$$c_{1d}(L,N) = L \cdot L \cdot w \cdot 2 = 2wL^2$$
$$c_{2d}(L,N) = L \cdot \frac{L}{\sqrt{N}} \cdot w \cdot (2+2) = 4wL^2N^{\frac{1}{2}}$$
$$c_{3d}(L,N) = \frac{L^2}{\sqrt[3]{N}} \cdot \frac{L^2}{\sqrt[3]{N}} \cdot w \cdot (2+2+2) = 6wL^2N^{-\frac{2}{3}}$$

Dove $C(L,N)$ è il massimo volume di dati bidirezionali scambiati su un network link, L la dimensione di un lato di un reticolo cubico su cui si sta effetuando l'analisi, N il numero di processi, e w la dimensione di un singolo elemento del reticolo. Si evince chiaramente che la configurazione 3D è chiaramente quella ottimale in termini communication overhead in un contesto di strong scalability in quanto $C(L,N)$ dipende da $N^{-\frac{2}{3}}$. Bisogna sottolineare però che all'aumentare del numero di processi aumenterà il rapporto superficie e volume di ogni subdomain rendendo il volume di messaggi scambiati esiguo e rendendo il problema fortemente dominato dalla latenza. Ad ogni modo le considerazioni appena esposte si applicano minimamente al programma in questione in quanto non vi è comunicazione fra i processi e dunque la scelta della topologia si dimostra irrilevante o dipendente da caratteristiche intriseche del programma e non legate a overhead del protocollo di comunicazione o speficifiche tecniche del network utilizzato. Di seguito i tempi in secondi dell'esecuzione del programma in questione al variare della distribuzione dei processi lungo gli assi, come si può notare i risultati sono sostanzialmente simili.

Partizione	2400·100·100	800·300·100	1200·200·100
(24,1,1)	25.49514	/	/
(8,3,1)	/	28.6845	/
(12,2,1)	27.82	/	28.8654
(6,4,1)	28.2155	/	27.8347
(4,6,1)	/	27.31	/
(4,3,2)	/	26.5097	
(3,4,2)	27.1821	/	24.2257
(6,2,2)	27.908	/	26.7531

3 Ping Pong Benchmark

Intel MPI Ping Pong La seguente analisi di performance per point-to-point communication è stata ottenuta utilizzando il benchmark IMB PingPong di Intel® inoltre il programma è stato eseguito variando la topologia dei process, il protocollo di rete e la dimensione del messaggio scambiato.
Per modellare la larghezza di banda e il tempo di esecuzione è stata utilizzata la seguente formula:

$$T = T_l + \frac{N}{B} \tag{1}$$

Con N il numero di bytes del messaggio, B la *bandwidth* asintotica del network e con T_l la latenza.
Il modello rappresenta chiaramente una semplificazione dell'effettiva relazione fra banda, latenza e dimensione del messaggio e per questo motivo presenta diverse criticità. In particolare per quanto riguarda la latenza:

1. Tutti i protocolli di trasmissione dati hanno un *overhead* dovuto a ragiononi amministrative proprie del protocollo

2. Alcuni protocolli (come, ad esempio, TCP/IP utilizzato su Ethernet) definiscono una dimensione minima del messaggio, quindi anche se l'applicazione invia un singolo byte, un piccolo "frame" di $N > 1$ byte viene comunque trasmesso.

3. Inizializzare unoscambio di messaggi è un processo complicato che coinvolge più livelli di software, a seconda della complessità del protocollo. Ognuno di questi layers di software aumentano la latenza.

Inoltre bisogna ricordarsi la larghezza del messaggio varia in genere attraverso 8 ordini di grandezza mentre la larghezza di banda è al meno di 3 ordini di grandezza inferiori rispetto i messaggi di dimensione maggiore.
Nell'immagine è stata riportata a titolo di esempio la curva effettiva della banda in funzione di N e quella invece ottenuta utilizzando il modello.

Come si può notare il modello non riesce a eseguire un fit adeguato per quanto riguarda i valori intermedi di N e la ragione di questa scarsa capacità rappresentativa può essere dovuta differenti fattori. Molti protocolli per esempio cambiano algoritmi di buffer a seconda della dimensione del messaggio oppure per gestire messaggi di dimensione elevata ne effettuano una divisione in *chunks* più piccoli.
Un ultimo dettaglio fondamentale rispetto al grafico dellì'andamento della *bandiwidth* riguarda il ruolo della cache e della dimensione del messaggio scambiato. Nel caso infatti di processi eseguiti su core adiacenti, quindi all'interno dello stesso socket, si può beneficiare della condivisione della L3 e dunque si possono raggiungere picchi più altri rispetto a quando si esegue Ping Pong su configurazione intra-socket e intra-node. Nel secondo caso infatti lo scambio di messaggi avviene attraverso la memoria principale andando ad influire così sulla latenza e diminuendo il valore massimo di latenza, è interessante notare però che il grafico mantiene un andamento sostanzialmente simile ovvero in tutti e tre i casi i valori di banda crollano immediatamente una volta raggoiunto il loro picco. La ragione di questo comportamento dipende dall'implentazione del benchmark da parte di Intel® .
Andando ad analizzare il sorgente è possibile notare infatti che il programma esegue un numero di ripetizioni proporzionale alla dimensione del messaggio, per tale motivo il trasferimento del sendbuffer0 (cioè il sendbuffer associato alla prima iterazione) dal processo 0 al receiving buffer0 del processo 1 può essere implementato come operazione single-copy da parte del destinatario. Se le dimensioni del messaggio sono sufficientemente ridotte, i dati del sendbuffer0 si trovano nella cache del processo 1 e non è necessario sostituirli o modificarli a meno che il sendbuffer0, non venga modificato cosa che non si verifica, stessa cosa poi si verifica con sendbuffer1 inviato dal processo 1 al processo 0. Alla fine della prima iterazione dunque entrambi i processi avranno nella cache uno il sendbuffer dell'altro e le seguenti iterazioni dell'algoritmo saranno *in-cache operations*. Quando le dimensioni del messaggio sono troppo grandi perchè si possa salvare nella cache sia receivebuffer che sendbuffer le operazioni di scambio fra i due processi non saranno più *in-cache operations* ma sarà necessario passare attravero la *main memory*.
Andiamo ad analizzare ora le performance nel caso si usi un'interfaccia Infiniband e una interfaccia Ethernet (Figure

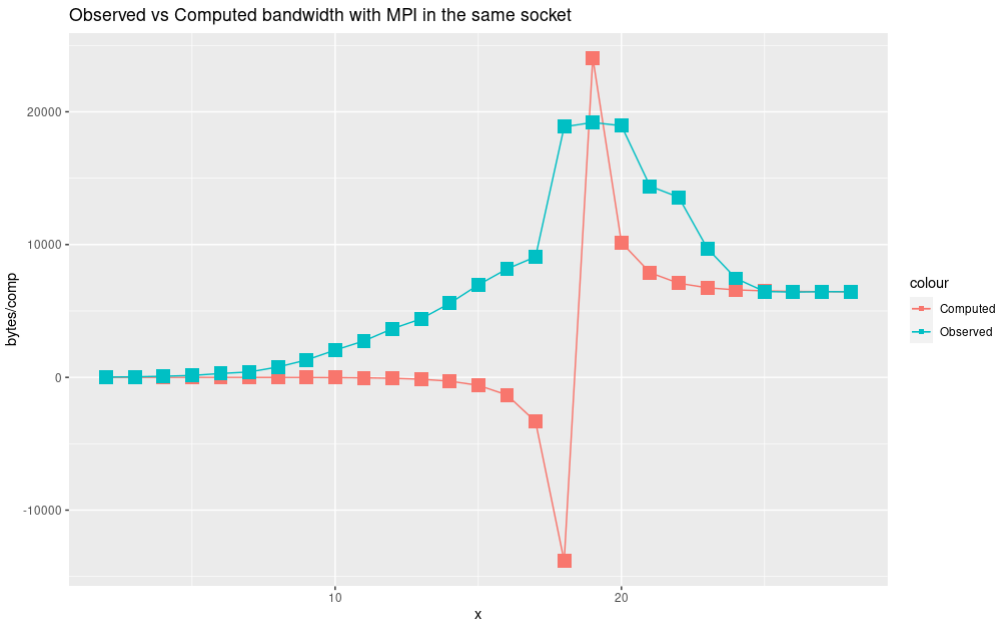

Figure 4: Observed vs Computed Bandwidth

4, Figure 5).
Si può notare immediatamente che le performance sono nettamente a favore di Infiniband, questo dovuto principalmente al fatto che implementa a differenza di Ethernet una configurazione RDMA. Ovvero è possibile accedere alla memoria su una macchina remota senza interferire con il lavoro della cpu su quel sistema, più nello specifico

1. Le applicazione possono realizzare un scambio dati senza il coinvolgimento del network software stack. I dati sono inviati e ricevuti direttamente sui rispettivi buffer senza venire copiati fra i vari network layers.

2. Kernel bypass - le applicazioni possono effettuare direttamente lo scambio di dati nel user-space senza coinvolgere il kernel.

3. Nessun coinvoglimento della CPU - le applicazioni possono accedere alla memoria remota senza interferire con la CPU nel server remoto. La memoria remota del server verrà letta senza alcun intervento da parte del processo (o processore) remoto. Inoltre, le cache della CPU remota non verranno riempite con il contenuto della memoria a cui si accede.

Un' altra importante differenza coniste nel fatto la dimensione minima di un frame nel protocollo ethernet è di 64 bytes questo ovviamente influirà negativamente sulla trasmissione di messagi di dimensione inferiore. In generale ad ogni modo l'overhead apportato da ethernet è maggiore poichè i software layers implementati dal protocollo sono più complessi rispetto quelli implementati da inifniband. Confrontando i risultati dei benchmark ottenuti con IntelMPI e MPI (figure 6 figure 7) è possibile notare che c'è un significativo vantaggio in termini di banda massima raggiunta a favore di MPI, nelle comunicazione inter-socket e in particolare in quelle intra-socket.

Si è poi ripetuto gli esperimenti su nodi GPU e le performance complessivamente sono rimaste invariate rispetto al caso thin node.

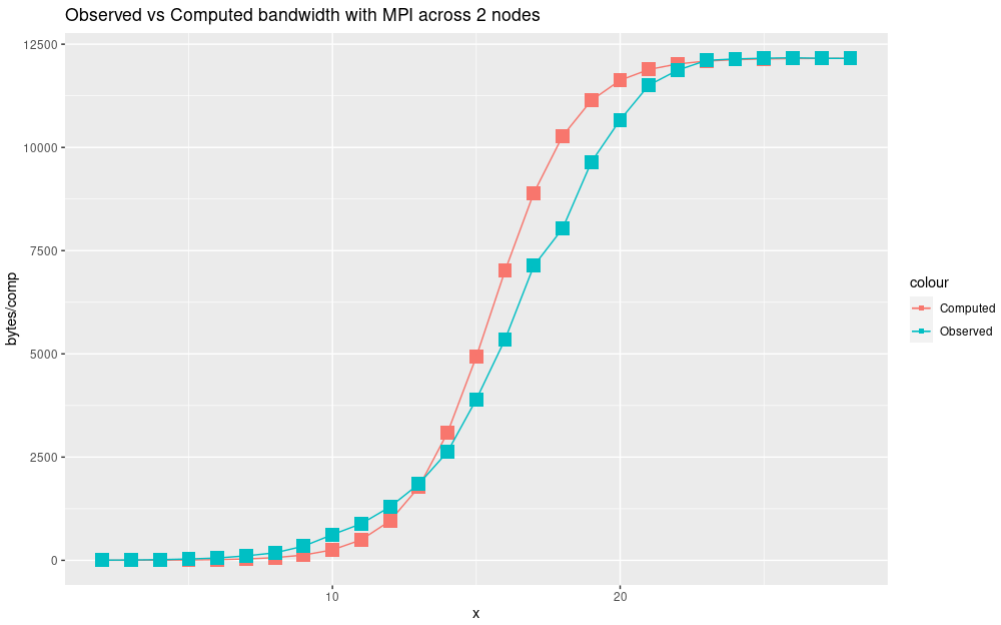

Figure 5: Observed vs Computed Bandwidth

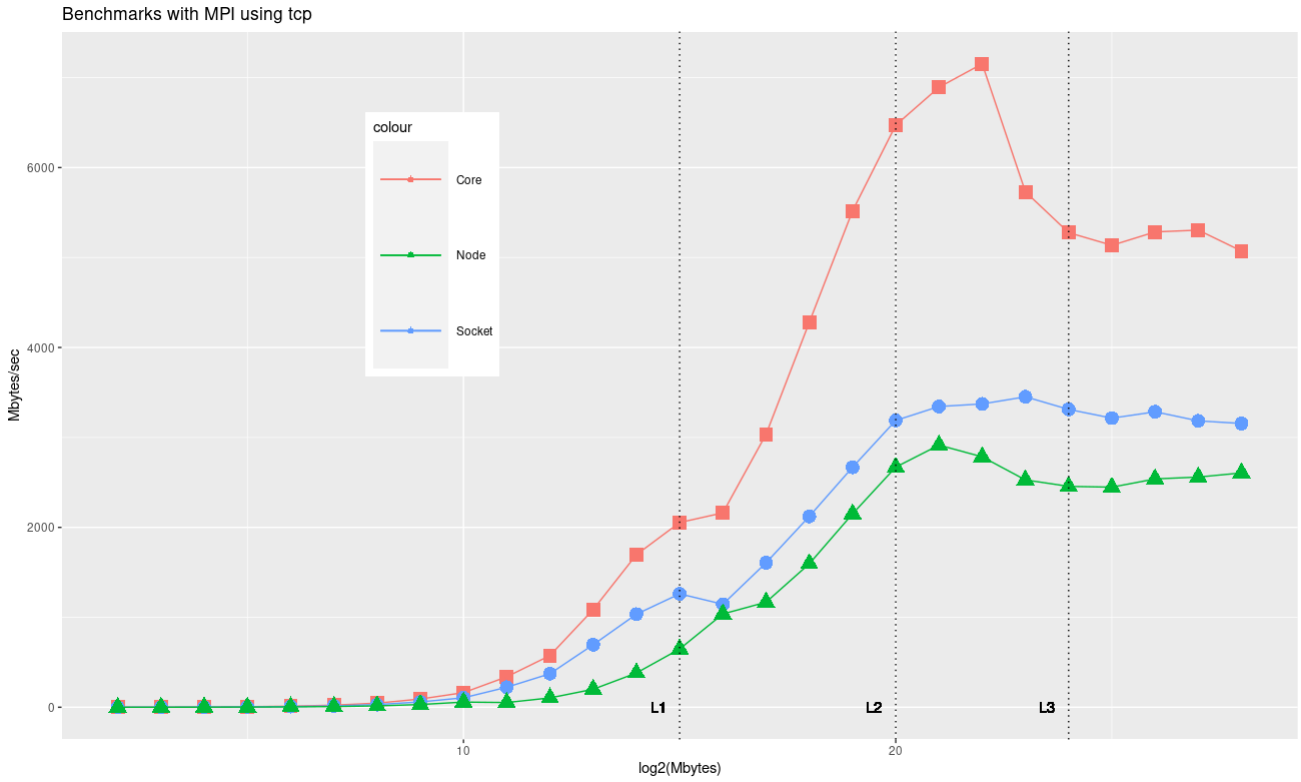

Figure 6: MPI using Ethernet

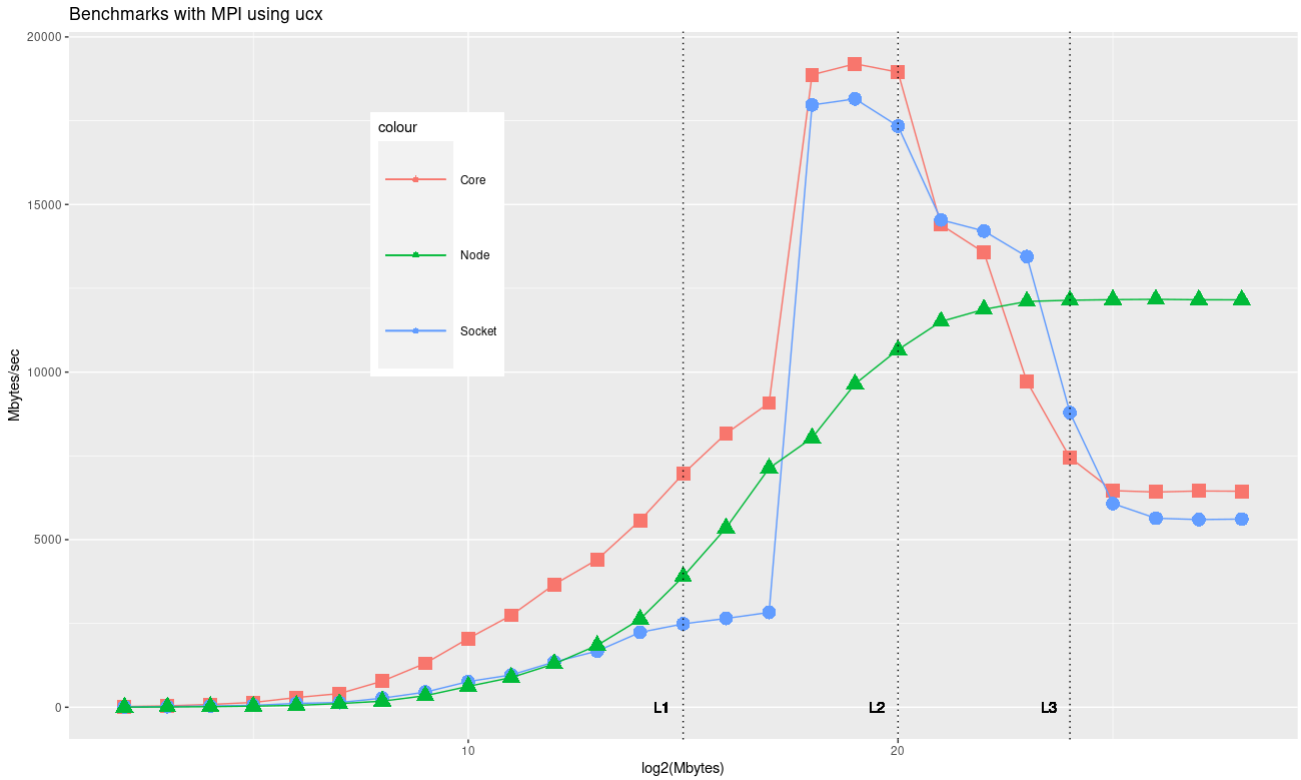

Figure 7: MPI using Infiniband

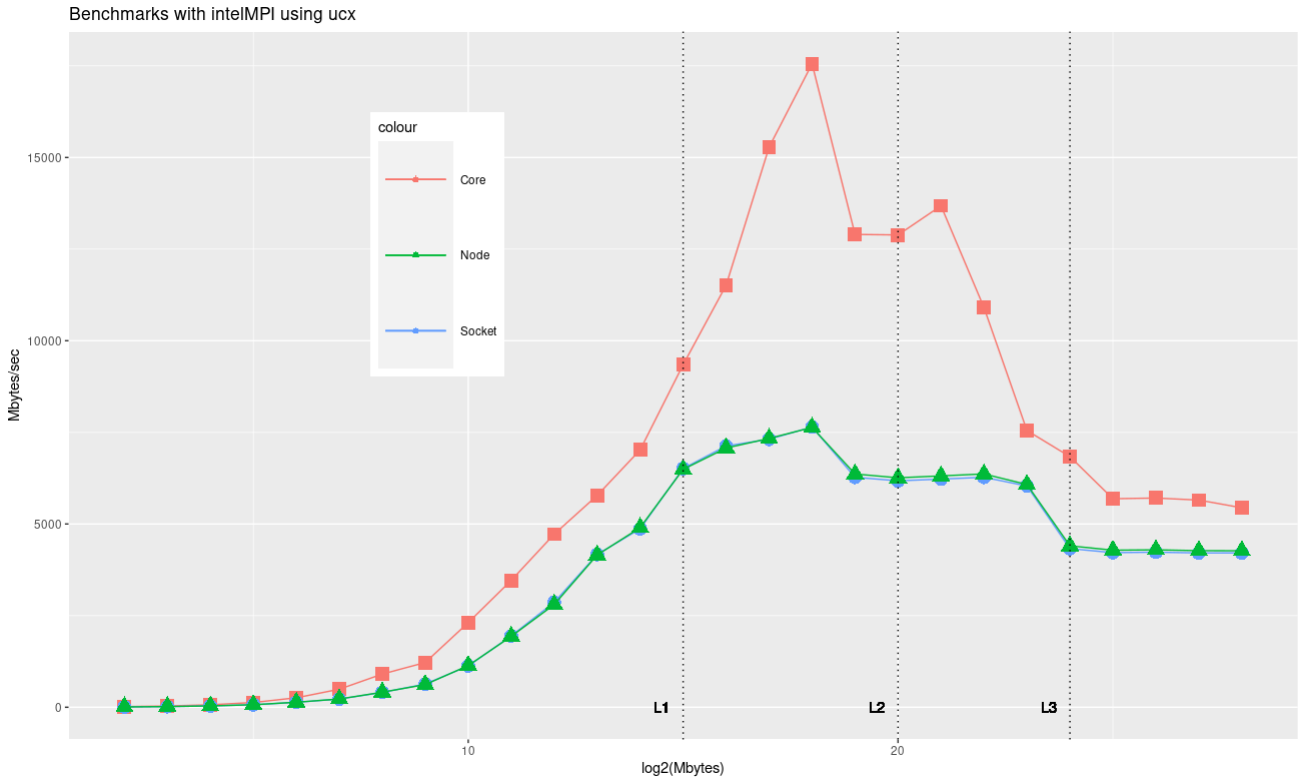

Figure 8: Intel MPI

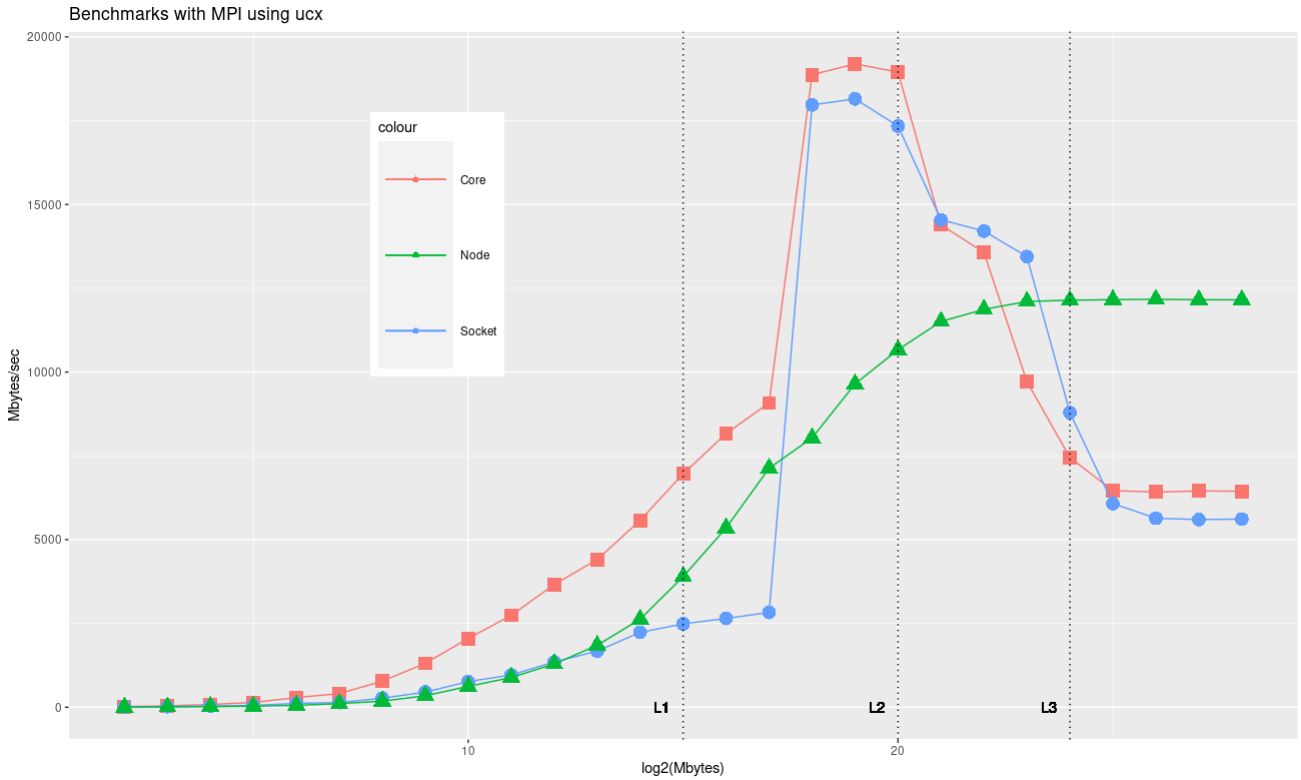
Benchmarks with MPI using ucx
20000
15000
10000
5000
0
Mbytes/sec
colour
Core
Node
Socket
L1
L2
L3
10
20
log2(Mbytes)

Figure 9: MPI

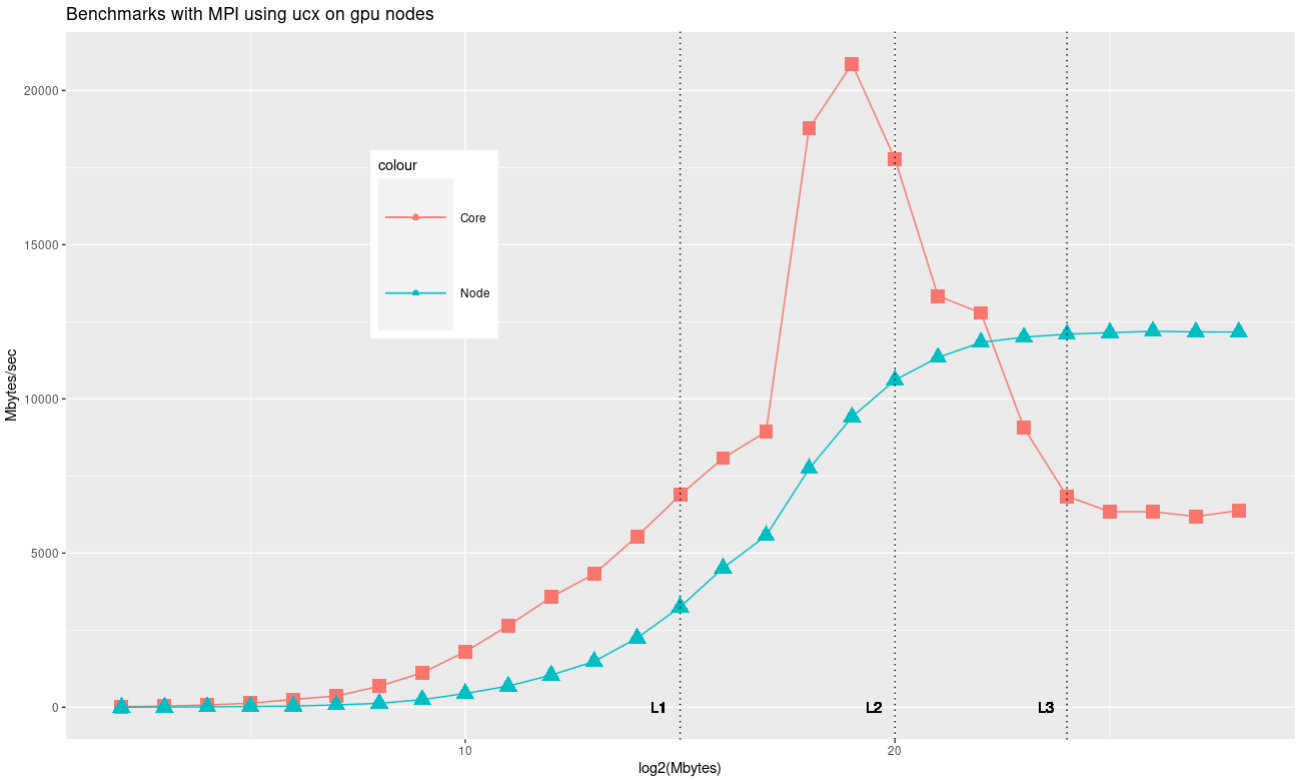

Figure 10: MPI using gpu nodes

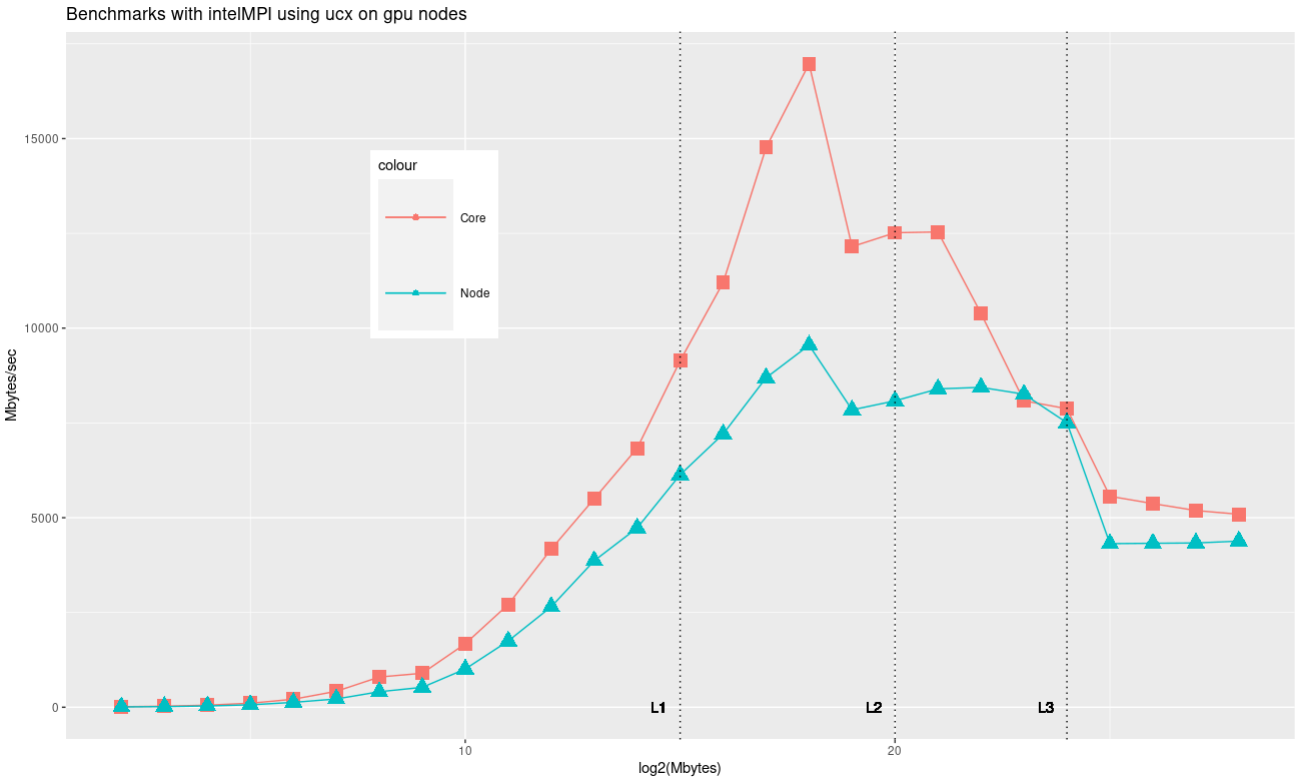

Figure 11: intel MPI using gpu nodes

4 Application Performance

Questa terza sezione è dedicata all'analisi delle performance e della scalabilità del programma Jacobi3D [1] al variare del numero e della topologia
dei processi coinvolti nell'esecuzione del programma. La formula utilizzata per studiare le performance del programma è:

$$P(L,N) = \frac{L^3 N}{T_s(L) + T_c(L,N)} \tag{2}$$

con L rappresenta la dimensione di un lato del cubo del reticolo e N il numero di processi. Il T_s nella formula rappresenta il tempo necessario per fare l'update di un subdomain la cui dimensione, in un contesto di weak scaling, rimane costante al variare dei processi impiegati e dunque anche T_s rimane sostanzialmente costante. Il T_c rappresenta il tempo di comunicazione ovvero il tempo impiegato dai processi per lo scambio degli halo layers, queste operazioni dipendono dal numero e dalla dimensione dei *domain cuts*, quindi sono influenzati dal numero di processi coinvolti. Il comportamento di $T_c(L,N)$ è descritto infatti dalla seguente formula:

$$T_c(L,N) = \frac{c(L,N)}{B} + k\lambda \tag{3}$$

Dove k rappresenta il numero di assi lungo il numero di processi coinvolti è maggiore di 1, mentre $c(L,N)$ è il volume dei dati scambiati ed è espresso come:

$$c(L,N) = L^2 \cdot k \cdot 2 \cdot 8 \tag{4}$$

Si può notare quindi che il massimo valore che può assumere k è 6 inoltre fissta una topologia dei processori e un protocollo per lo scambio di dati il valore di T_c dipenderà solo da k in quanto B e λ sono costanti. Per il calcolo della $T_c(L,N)$ si è scelto i valori di banda e latenza ricavati esercizio precedente. Per studiare il weak scaling si è scelto come dimensione del subdomain un reticolo di lato $L = 500$ e i boundaries del domain periodici, con tale configurazione si è il seguente output:

Performance per un singolo core su THIN node `average time = 1.12343`
`average jacobi time = 1.092959`
`average MLUPs = 111.2664`

[1] "https://blogs.fau.de/hager/hpc-booksamplecode"

type	perf	computed perf	penalty
thin	111.2664	114.3685	1
4ss	444.9474	457.4574	1.0002653
8ss	874.0444	905.6723	1.0184047
12ss	1313.4709	1357.174	1.0165406
4ds	439.8586	451.8131	1.0118375
8ds	878.0999	905.5415	1.0137012
12ds	1317.1498	1358.3122	1.0137012
12n	1338.0808	1368.9472	0.9978444
24n	2449.2595	2519.8686	1.0902858
48n	5173.5053	5323.9621	1.032334

type	$C(L, N)$	T_c	cT_c
thin	0	0.03047146	0
4ss	16	0.03297821	0.00224776
8ss	24	0.04332604	0.00337164
12ss	24	0.04014622	0.00337164
4ds	16	0.03232436	0.00224776
8ds	24	0.03788256	0.00337164
12ds	24	0.03788256	0.00337164
12n	24	0.02864767	0.00337164
24n	24	0.03784733	0.0035256
48n	24	0.04390709	0.01113206

Si è ripetuto poi gli stessi esperimenti su nodi GPU, le performance per un singolo core sono:

```
average time = 1.799005
average jacobi time = 1.744283
average MLUPs = 69.62272
```

type	computed perf	perf	penality
thin	71.66267	69.48286	1
4ss	268.61393	262.43308	1.0590565
8ss	497.35701	483.05095	1.1507335
12ss	699.4611	679.49083	1.227087
4ds	260.73949	254.94896	1.0901455
8ds	493.51905	479.28571	1.1597736
12ds	722.74249	700.88416	1.1896322
12n	913.90592	887.06642	0.9399458
24n	1613.04537	1570.05691	1.0621199
48n	5367.43619	5173.50534	0.6446649

type	cln	tc	ctc
thin	0	0.05472136	0
4ss	16	0.04620893	0.002368837
8ss	24	0.06310011	0.003553256
12ss	24	0.06658045	0.003553256
4ds	16	0.04592282	0.002368837
8ds	24	0.0637272	0.003553256
12ds	24	0.0682792	0.003553256
12n	24	0.05321342	0.003553256
24n	24	0.05483971	0.003917037
48n	24	0.04390709	0.002003974

La quarta colonna rappresenta quanto la crescita delle performance sia rallentata rispetto ad un modello in cui si verifica la scalabilità perfetta. In tale configurazione infatti $N \cdot P_1(L, N) = P(L, N)$ ovvero è possibile parallelizzare l'intero volume del lavoro. Possiamo notare che i valori teorici di $T_c(L, N)$ non riescono a predirre efficacemente i valori osservati; una ragione dell'inefficacia del modello dipende dal fatto che le operazioni di send and receive effettuate nello scambio degli halo layers possono sovrapporsi lungo ognuno delle 6 direzioni, perciò bisognerebbe considerare valori asintotici di bandwidth per uno scambio dati *full-duplex* mentre il benchmark di Intel® Ping Pong stimano latenza e banda per uno scambio *half-duplex*. Si sono poi ripetuti i medesimi esperimenti in un contesto di strong scaling fissando la dimensione di un lato della matrice a $L = 1200$

type	cperf	perf	penalty
thin	114.8680	113.4620	1.0000000
4ss	459.0551	452.5394	1.0028917
8ss	906.4822	886.2511	1.0241972
12ss	1340.3489	1305.2428	1.0431346
4ds	458.0264	451.6331	1.0049041
8ds	1366.8973	1335.6252	0.6796038
12ds	910.8814	890.2755	1.5293512
12n	1369.0482	1336.4235	1.0187968
24n	2558.1389	2486.3643	1.0952087
48n	5390.8712	5130.5998	1.0615086

type	$C(L, N)$	tc	ctc
thin	0.00000	0.18641069	0.0000000000
4ss	36.57830	0.05981501	0.0056166191
8ss	34.56000	0.04873568	0.0052198934
12ss	26.37338	0.03860290	0.0039278361
4ds	36.57830	0.05902268	0.0056166191
8ds	34.56000	0.03481905	0.0052198934
12ds	26.37338	0.04783636	0.0039278361
12n	26.37338	0.03474035	0.0039278361
24n	16.61338	0.02206673	0.0025671295
48n	10.45440	0.01715213	0.0008912807

Si può notare che anche in questo caso la scabilità è soddisfacente così come il modello usato per stimare le performance.