

MM MPI

Diego Russo

Chi sono

- Diego Russo
- Studente Specialistica, 231423
- Sviluppatore Software @ ARM Ltd
- Vivo e lavoro a Cambridge, UK
- <http://www.diegor.uk> - me@diegor.it
- @diegor

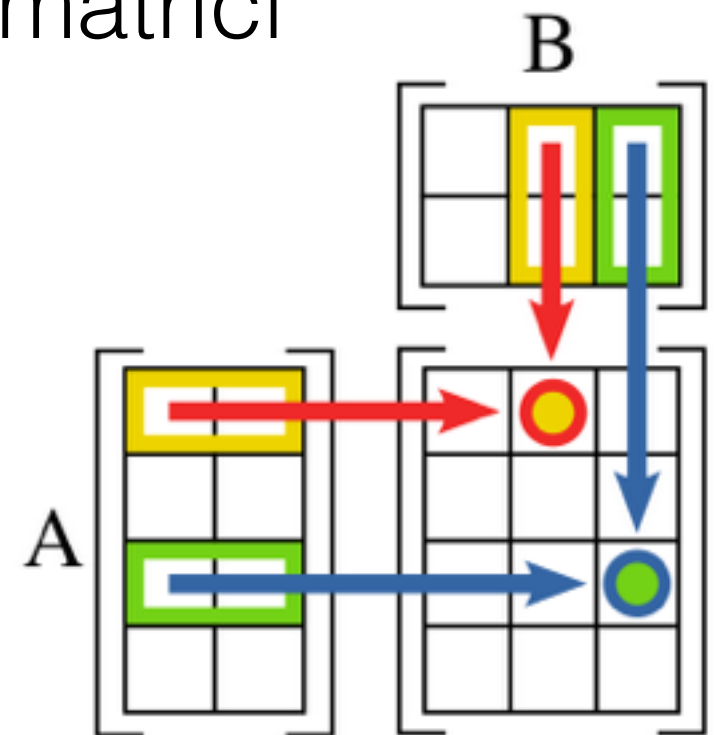


Il progetto

- Sviluppo ed implementazione di un codice parallelo per il calcolo del prodotto matrice-matrice
- L'algoritmo deve utilizzare la decomposizione a blocchi per le matrici e fare uso di primitive collettive/virtuali
- In particolare, lo studente deve sviluppare un proprio algoritmo parallelo non triviale e valutarne speedup ed efficienza.

La moltiplicazione tra matrici

- Prodotto tra righe e colonne tra due matrici
- Diverse tipologie di matrici
- # colonne di A = # di righe di B
- $A[4 \times 2] \times B[2 \times 3] = C[4 \times 3]$



$$c_{ij} = \sum_{r=1}^n a_{ir}b_{rj} = a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{in}b_{nj}$$

Cannon

A

A(0,0)	A(0,1)	A(0,2)
A(1,0)	A(1,1)	A(1,2)
A(2,0)	A(2,1)	A(2,2)

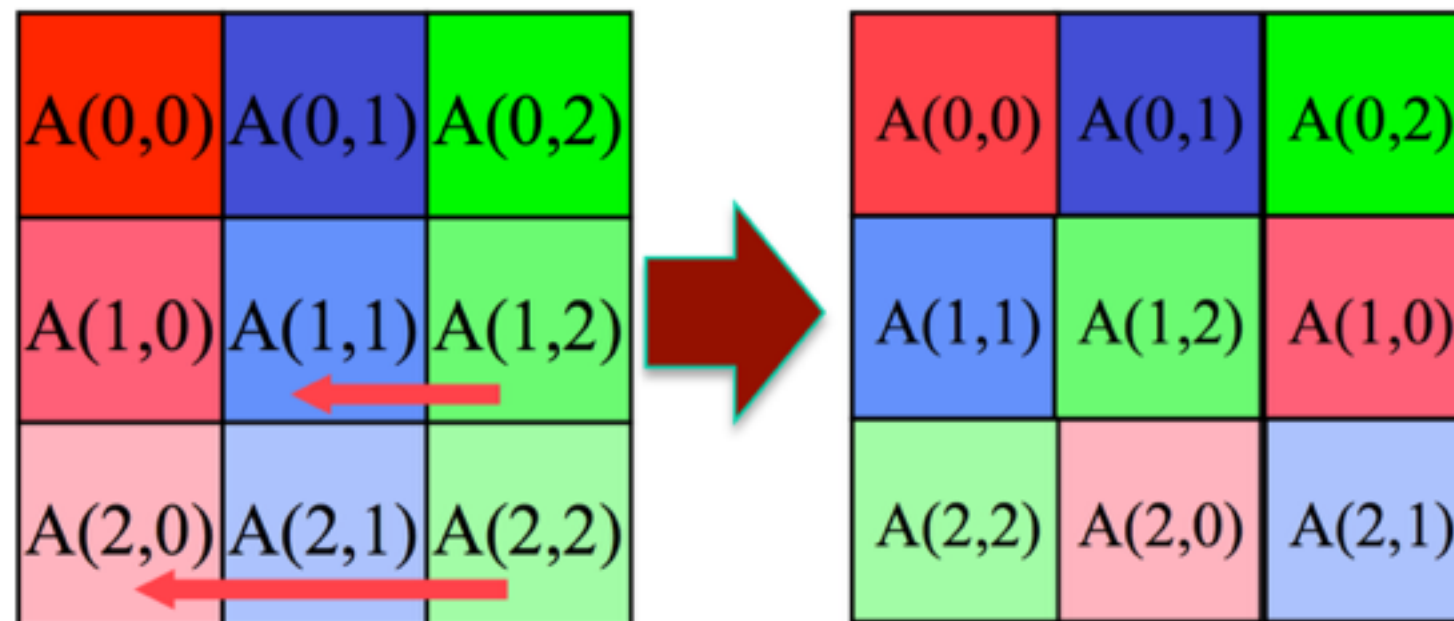
B

B(0,0)	B(0,1)	B(0,2)
B(1,0)	B(1,1)	B(1,2)
B(2,0)	B(2,1)	B(2,2)

$$C[1,2] = A[1,0] * B[0,2] + A[1,1] * B[1,2] + A[1,2] * B[2,2]$$

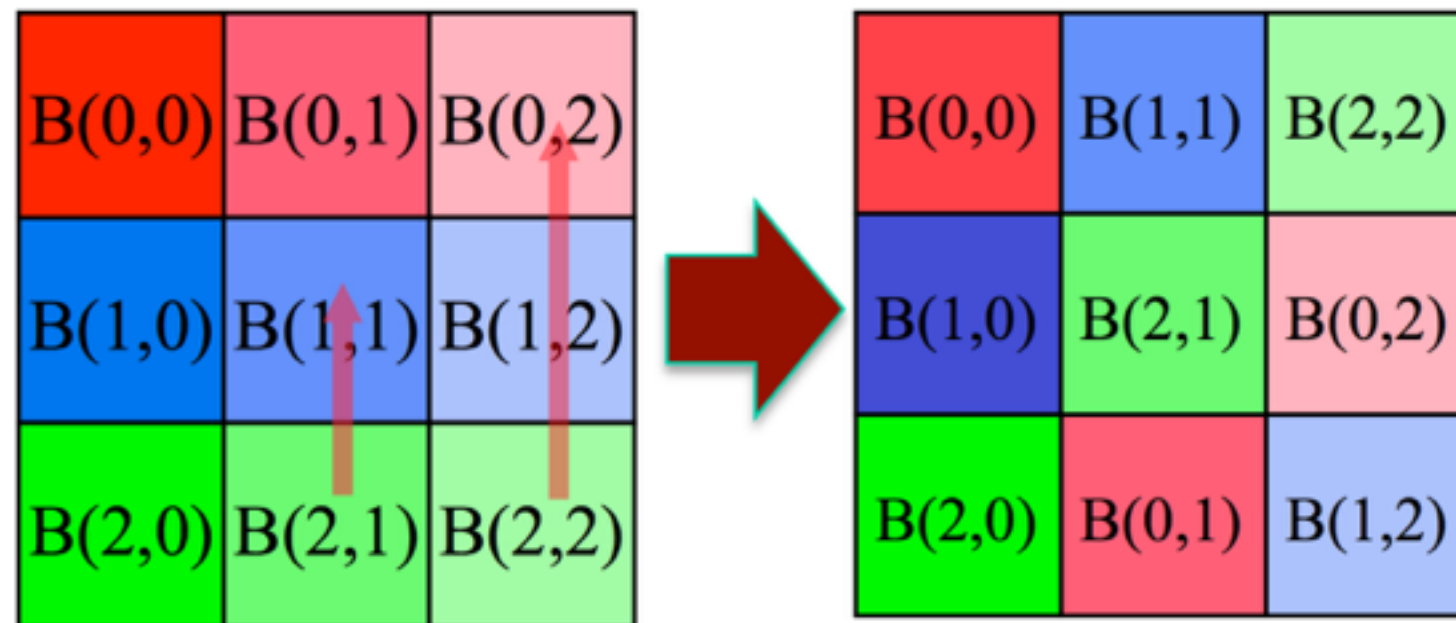
- Partizione in p blocchi quadrati
- Ogni processo ha un blocco n / \sqrt{p}
- Dati inviati in maniera incrementale in \sqrt{p} passi

Cannon



Shift Iniziale di A : left di i step su ogni riga

Cannon



Shift Iniziale di B : up di j step su ogni colonna

Cannon

A(0,1)	A(0,2)	A(0,0)
A(1,2)	A(1,0)	A(1,1)
A(2,0)	A(2,1)	A(2,2)



B(1,0)	B(2,1)	B(0,2)
B(2,0)	B(0,1)	B(1,2)
B(0,0)	B(1,1)	B(2,2)



$$T_p = \frac{n^3}{p} + 2\sqrt{p}\left(t_s + \frac{t_w n^2}{p}\right) + 2\left(t_s + \frac{t_w n^2}{p}\right)$$

- Costo moltiplicazione di \sqrt{p} matrici $(n/\sqrt{p})^3$
- Costo di \sqrt{p} shift
- Costo shift iniziale

Ulteriore shift di A e B

MM MPI: il codice

- Scritto interamente in C
- Strutturato in moduli: serial, cannon, shared
- Ben documentato: commenti in ogni funzione
- Makefile everywhere: diverse versioni da compilare
- libreria MPI mvapich2
- Compilatore: gcc

```
contributors.txt  
data  
doc  
logs  
Makefile  
Makefile.mm.dev  
Makefile.mm.inc -> Makefile.mm.pg  
Makefile.mm.pg  
README  
run_mm_mpi_dev.sh  
run_mm_mpi_pg.sh  
src  
test.sh
```

MM MPI: src/

```
1 format.c
2 format.h
3 reset.c
4 reset.h
5 shared.h
6 utils.c
7 utils.h
```

src/shared/

```
1 check.c
2 gendat.c
3 Makefile
4 mm.c
5 mxm.c
```

src/serial/

```
1 check.c
2 gendat.c
3 Makefile
4 Makefile.cblas
5 Makefile.nonblock
6 Makefile.nonblock.cblas
7 Makefile.nonblock.openmp.inner
8 Makefile.nonblock.openmp.middle
9 Makefile.nonblock.openmp.nested
10 Makefile.nonblock.openmp.outer
11 Makefile.openmp.inner
12 Makefile.openmp.middle
13 Makefile.openmp.nested
14 Makefile.openmp.outer
15 mm.c
16 mxm.c
17 mxm-local.c
18 mxm-local.h
```

src/cannon/

MPI: primitive utilizzate

- MPI_Init, MPI_Finalize
- MPI_Comm_size, MPI_Comm_rank
- MPI_Barrier, MPI_Wtime
- MPI_Reduce
- MPI_Cart_create, MPI_Cart_coords, MPI_Cart_shift
- MPI_Sendrecv_replace
- MPI_Isend, MPI_Irecv, MPI_Waitall
- MPI_Comm_free

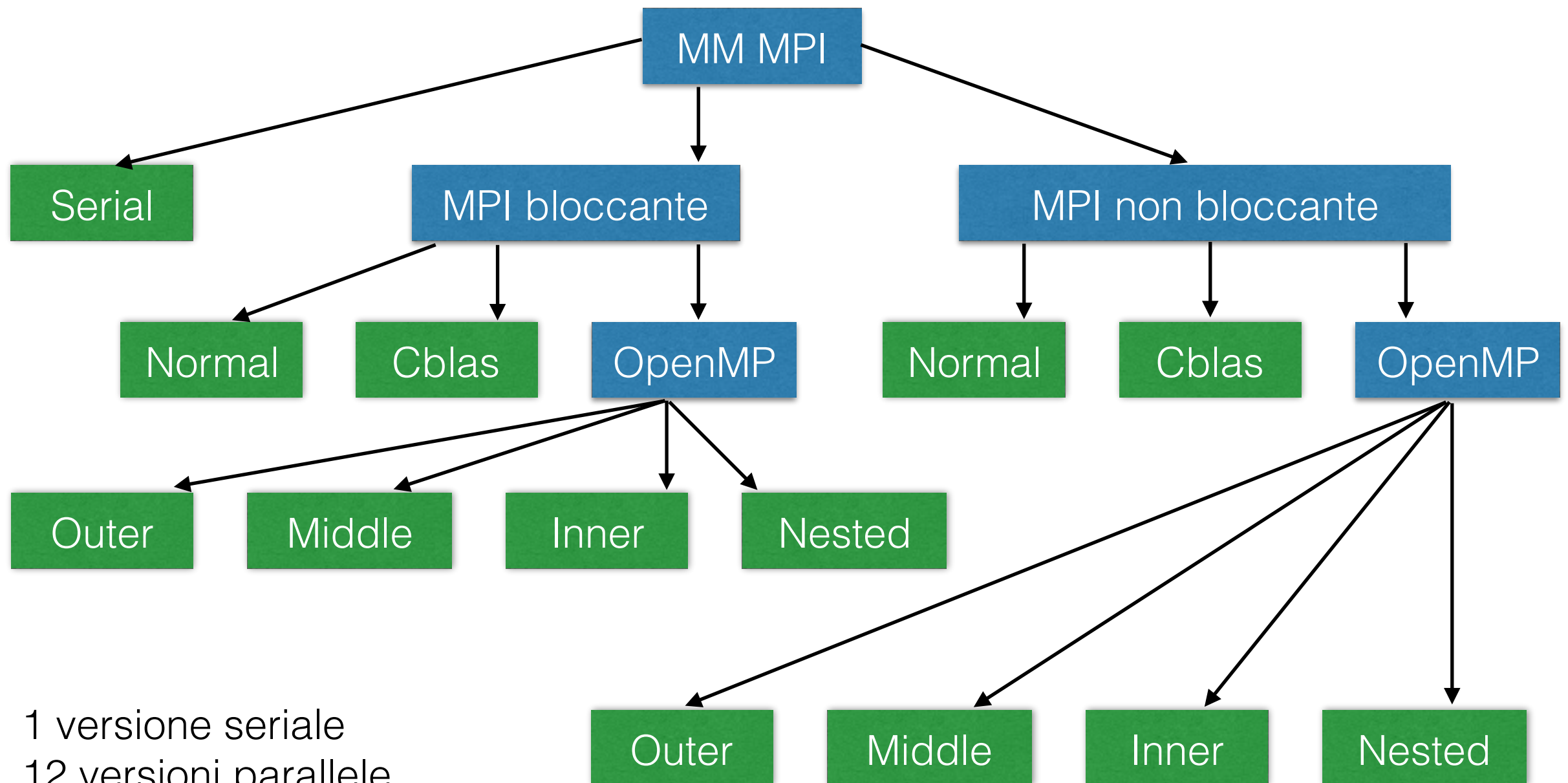
MM MPI: i Makefile

```
1 include ../../Makefile.mm.inc
2
3 VPATH = ../shared
4
5 OBJECTS = mm.o check.o gendat.o mxm.o mxm-local.o \
6           format.o reset.o utils.o
7
8 EXEC     = x.mm
9
10 ${EXEC}: clean ${OBJECTS}
11     @echo
12     ${LD_MPI} ${LDFLAGS} ${OBJECTS} ${LIBS} -o ${EXEC}
13
14 .SUFFIXES: .c .o
15 .c.o      :
16     @echo
17     ${CC_MPI} ${CFLAGS} -DCBLAS ${INCS} -c -o $@ $<
18
19 clean:
20     /bin/rm -f ${OBJECTS} ${EXEC}
```

Makefile.mm.inc

```
1 SELF_DIR := $(realpath .)
2
3 CC        = gcc-5
4 CC_MPI    = mpicc
5 INCS      = -I "$(SELF_DIR)/../shared/" -I /System/Library/Frameworks/
6             Accelerate.framework/Versions/Current/Frameworks/vecLib.framework/
7             Versions/Current/Headers/
8 CFLAGS    = -O3 -mmodel=medium -fopenmp
9 LD        = $(CC)
10 LD_MPI    = $(CC_MPI)
11 LDFLAGS   = $(CFLAGS) -lcblas
12 LIBS      = -L /System/Library/Frameworks/Accelerate.framework/Versions/
13             Current/Frameworks/vecLib.framework/Versions/Current/ -pthread
```

Ottimizzazioni



Generazione di matrici

1	2	3	4
1	2	3	4
1	2	3	4
1	2	3	4

A

1	1	1	1
0.5	0.5	0.5	0.5
0.33	0.33	0.33	0.33
0.25	0.25	0.25	0.25

B

```
a[i * L_DBLOCK + j] = (double)((coordinates[1] * L_DBLOCK) + j + 1);
```

```
b[i * N_DBLOCK + j] = 1.0 / (double)((coordinates[0] * L_DBLOCK) + i + 1);
```

Controllo del risultato

1	2	3	4
1	2	3	4
1	2	3	4
1	2	3	4

A

X

1	1	1	1
0.5	0.5	0.5	0.5
0.33	0.33	0.33	0.33
0.25	0.25	0.25	0.25

B

=

4	4	4	4
4	4	4	4
4	4	4	4
4	4	4	4

C

$$C[2,3] = A[2,0] * B[0,2] + A[2,1] * B[1,2] + A[2,2] * B[2,2] + A[2,3] * B[3,2]$$

$$C[2,3] = 1 * 1 + 2 * 0.5 + 3 * 0.33 + 4 * 0.25 = 4$$

MM MPI: esecuzione

Esecuzione seriale

```
1 ./x.mm_serial -f data/demo_data
```

Esecuzione con MPI su macchina singola

```
1 mpirun -n 16 ./x.mm_2D_cannon -f data/demo_data
```

```
1 $ cat data/2_repititions
2 4 4 4 2
3 8 8 8 2
4 16 16 16 2
5 32 32 32 2
6 64 64 64 2
7 128 128 128 2
8 192 192 192 2
9 256 256 256 2
10 384 384 384 2
11 512 512 512 2
12 768 768 768 2
13 1024 1024 1024 2
14 1536 1536 1536 2
15 2048 2048 2048 2
16 3072 3072 3072 2
17 4096 4096 4096 2
```

Input file

Esecuzione su ChemGrid

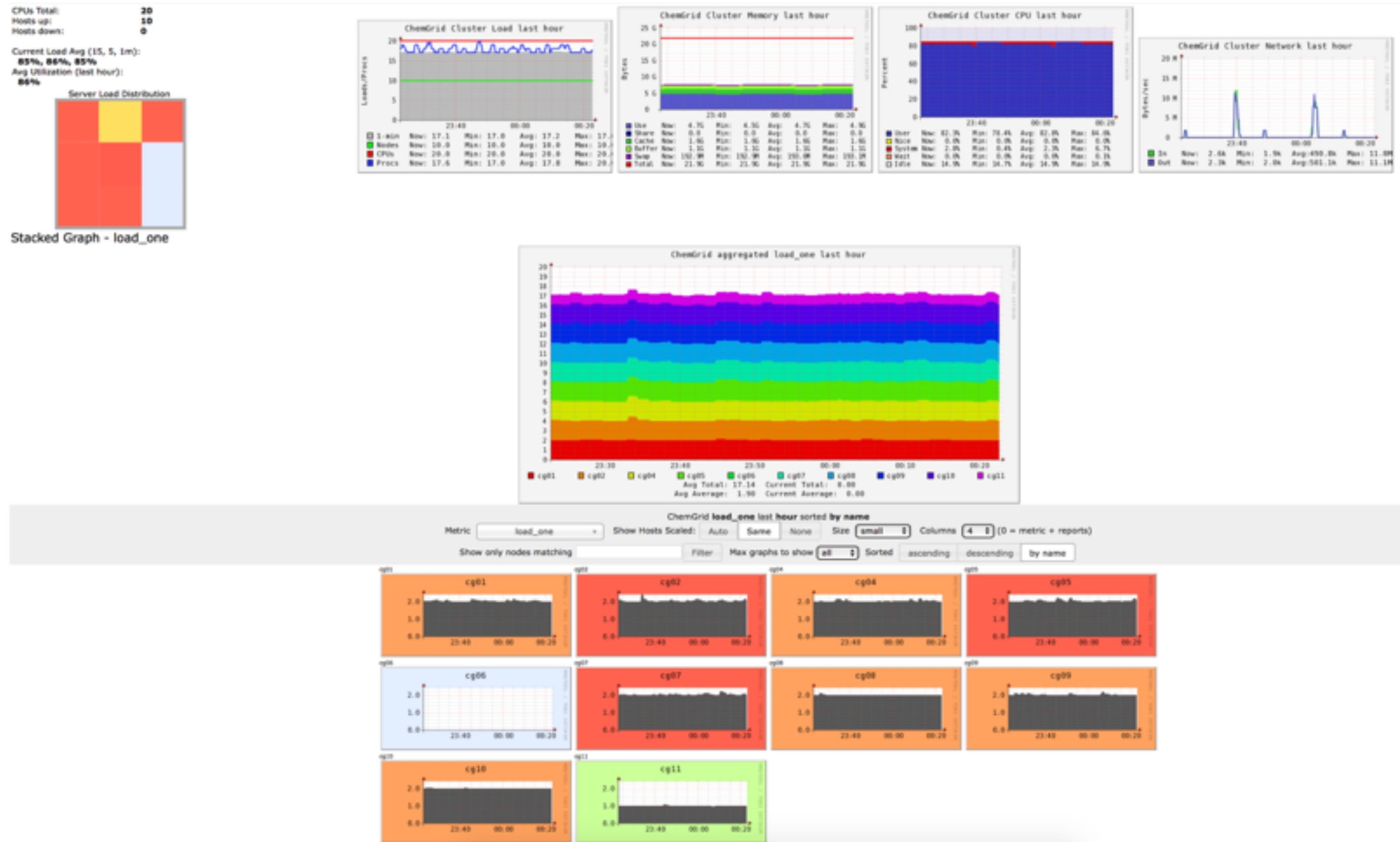
```
1 $ PROC_NAME=x.mm_2D_cannon
2 $ qsub -lnodes=8:ppn=2,mem=9050mb -d . -N $PROC_NAME -e logs -o logs -v EXE=
    $PROC_NAME,DATA=example_data run_mm_mpi_pg.sh

1 #!/bin/sh
2
3 mpirun_rsh -np $PBS_NP -hostfile $PBS_NODEFILE ./$EXE -f $DATA
```


Il cluster: ChemGrid

- 11 Nodi: 2 CPU 3GHz, 1GB RAM (4GB 3 nodi)
- 32 bit
- Network: 1000 Mbps, full duplex
- 3 nodi offline: cg03, cg06, cg11
- 8 nodi, 16 CPU disponibili
- 4, 9 16: quadrati perfetti, 9 non usabile
- $16 = 8 \text{ nodi} \times 2 \text{ CPU}$

II cluster

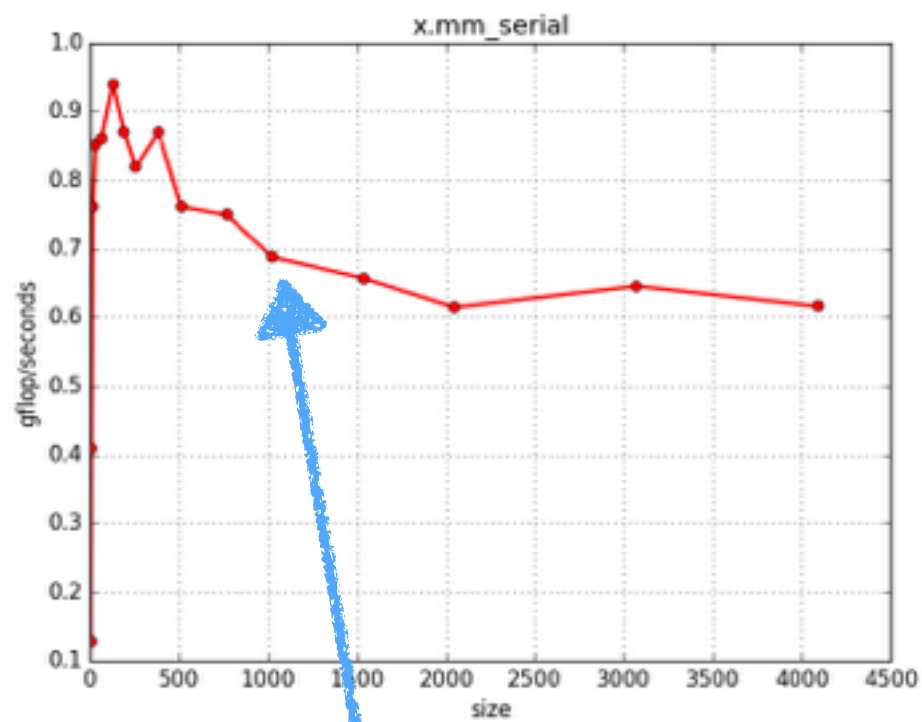


#> ssh -f -N -L 3000:cgcw.herla.unipg.it:80 diego@cgcw.herla.unipg.it

Esperimenti

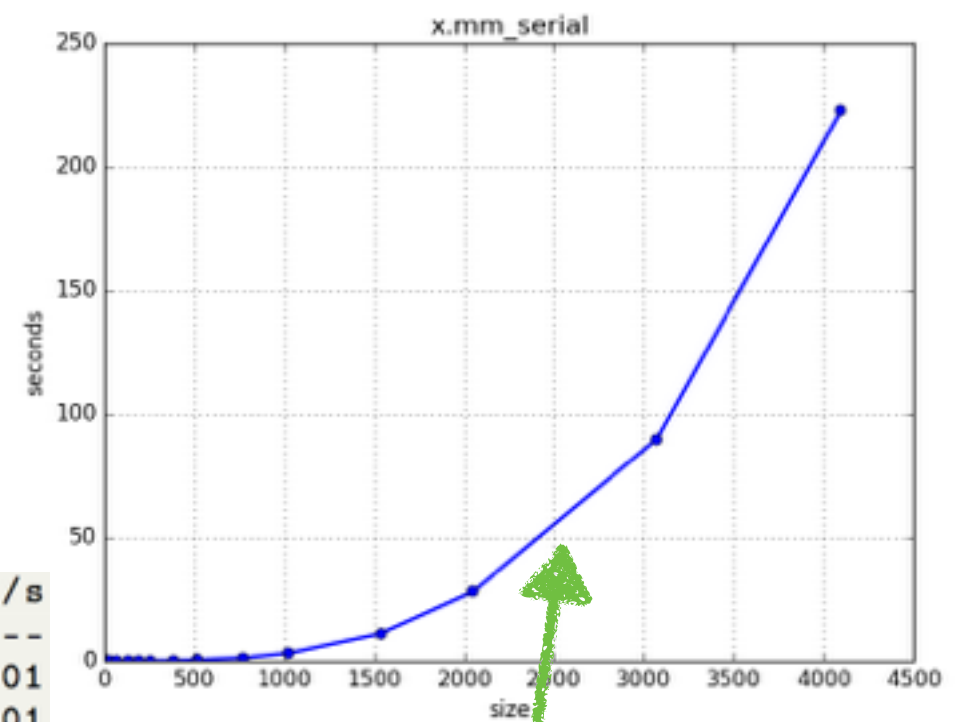
- Matrici quadrate di 4, 8, 16, 32, 64, 128, 192, 256, 384, 512, 768, 1024, 1536, 2048, 3072, 4096
- Non più grandi: computazioni troppo lunghe
- 2 ripetizioni per ogni dimensione
- $4096 * 4096 * 8 = 134217728$ bytes $\approx 131\text{MB}$
- $\text{Gflop/s} = 2 * n^3 / \text{time} * 1.0\text{e-9}$

Risultato seriale



Stabile

n	time	Gflops/s
4	0.0000	1.2800e-01
8	0.0000	4.0960e-01
16	0.0000	7.6205e-01
32	0.0001	8.5250e-01
64	0.0006	8.6170e-01
128	0.0045	9.4038e-01
192	0.0163	8.7050e-01
256	0.0409	8.1963e-01
384	0.1301	8.7073e-01
512	0.3523	7.6190e-01
768	1.2080	7.4996e-01
1024	3.1203	6.8823e-01
1536	11.0251	6.5739e-01
2048	27.9096	6.1555e-01
3072	89.7327	6.4616e-01
4096	222.7798	6.1693e-01

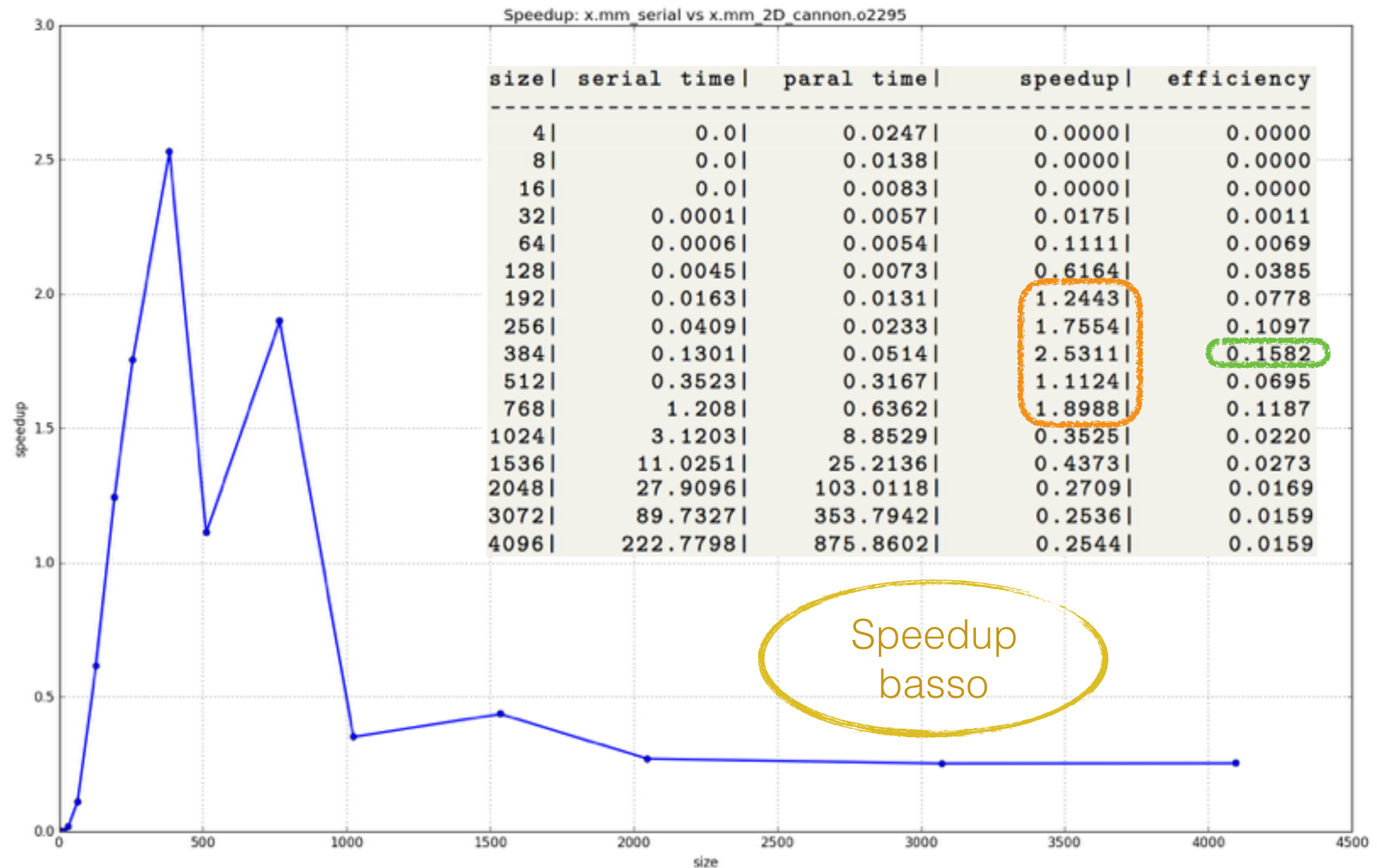


Crescita
esponenziale

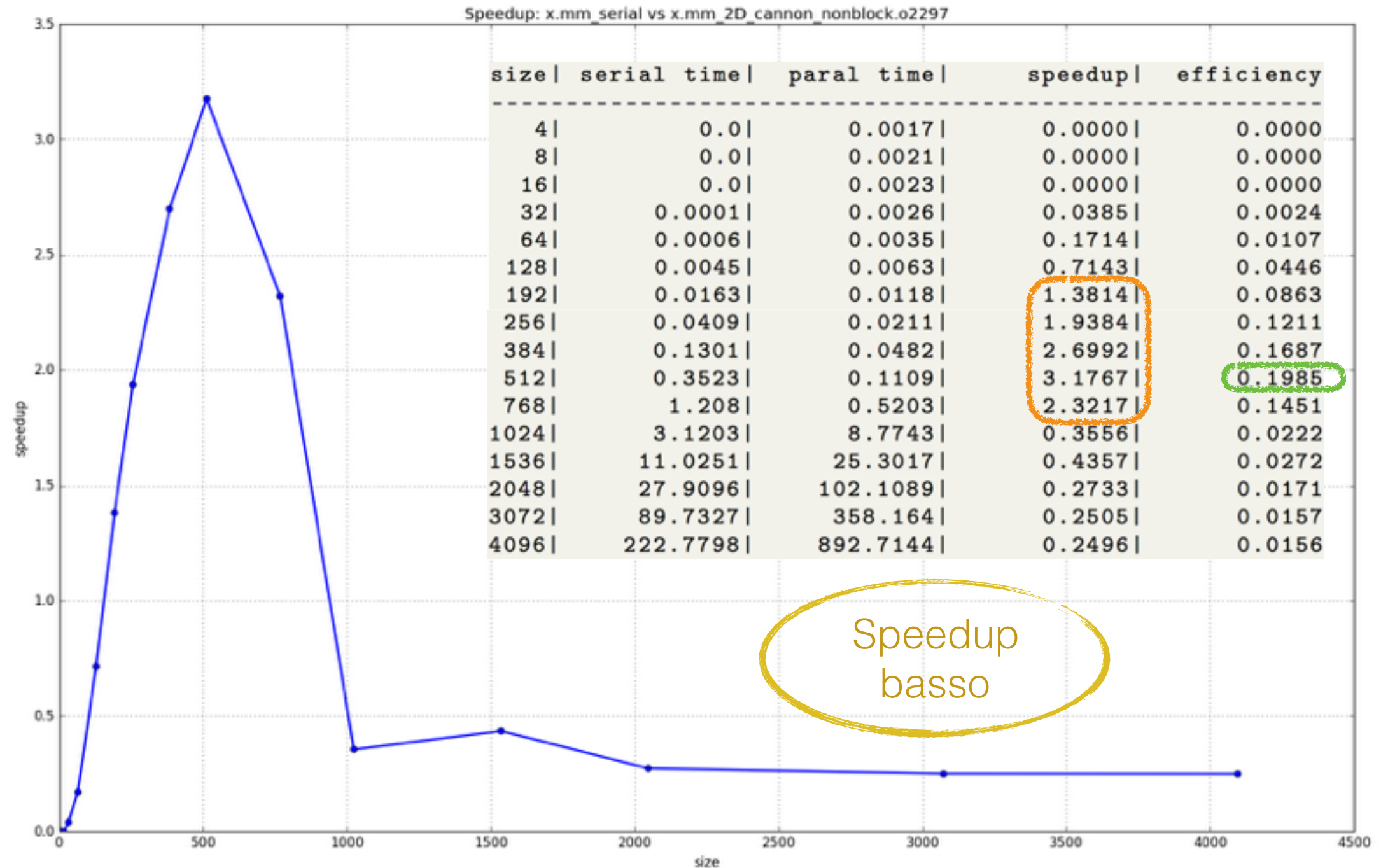
Speedup ed Efficienza

- $\text{Speedup} = T(1) / T(p)$
- Speedup ideale = p
- $\text{Efficienza} = S(p) / p$
- Efficienza ideale = 1

Risultato parallelo bloccante



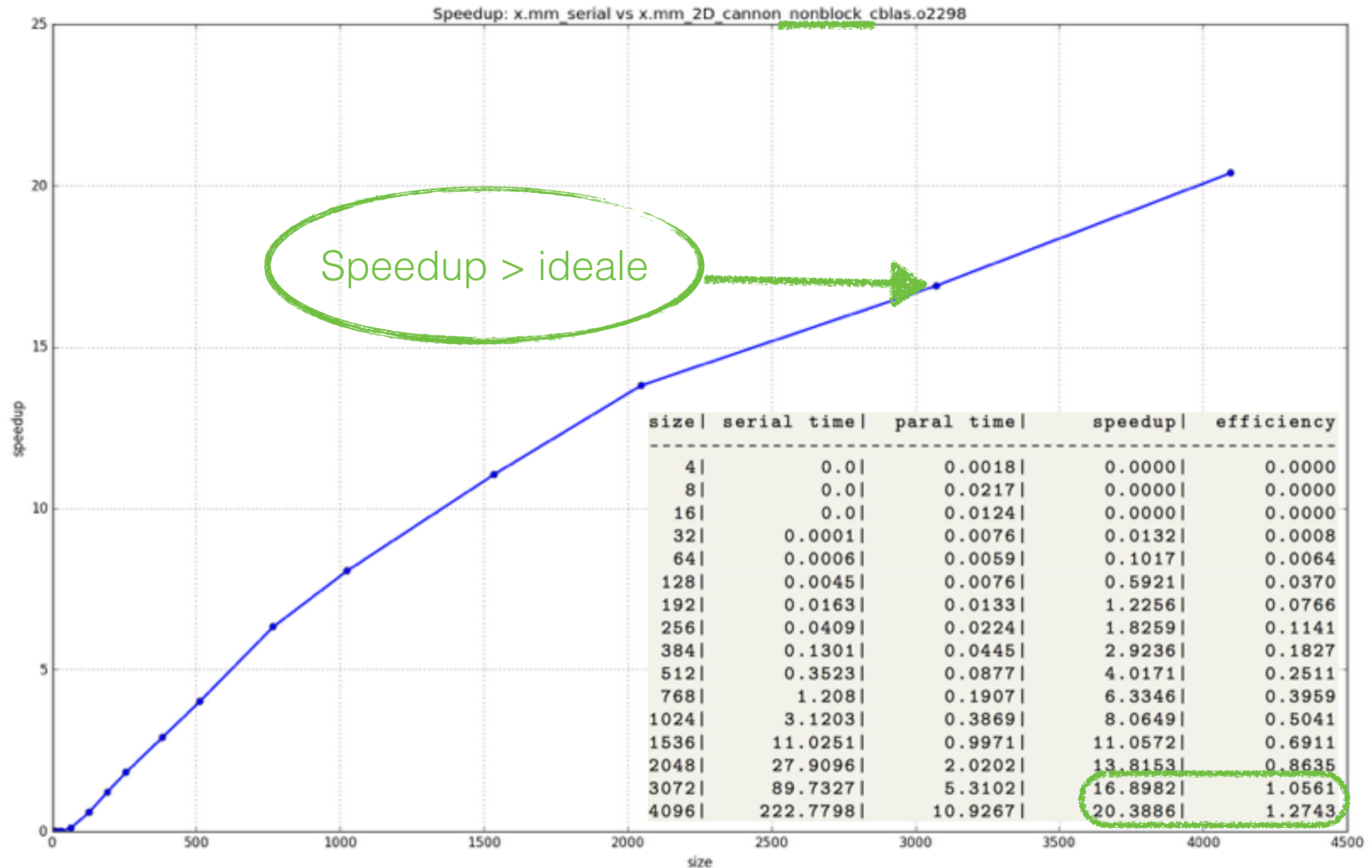
Risultato parallelo NON bloccante



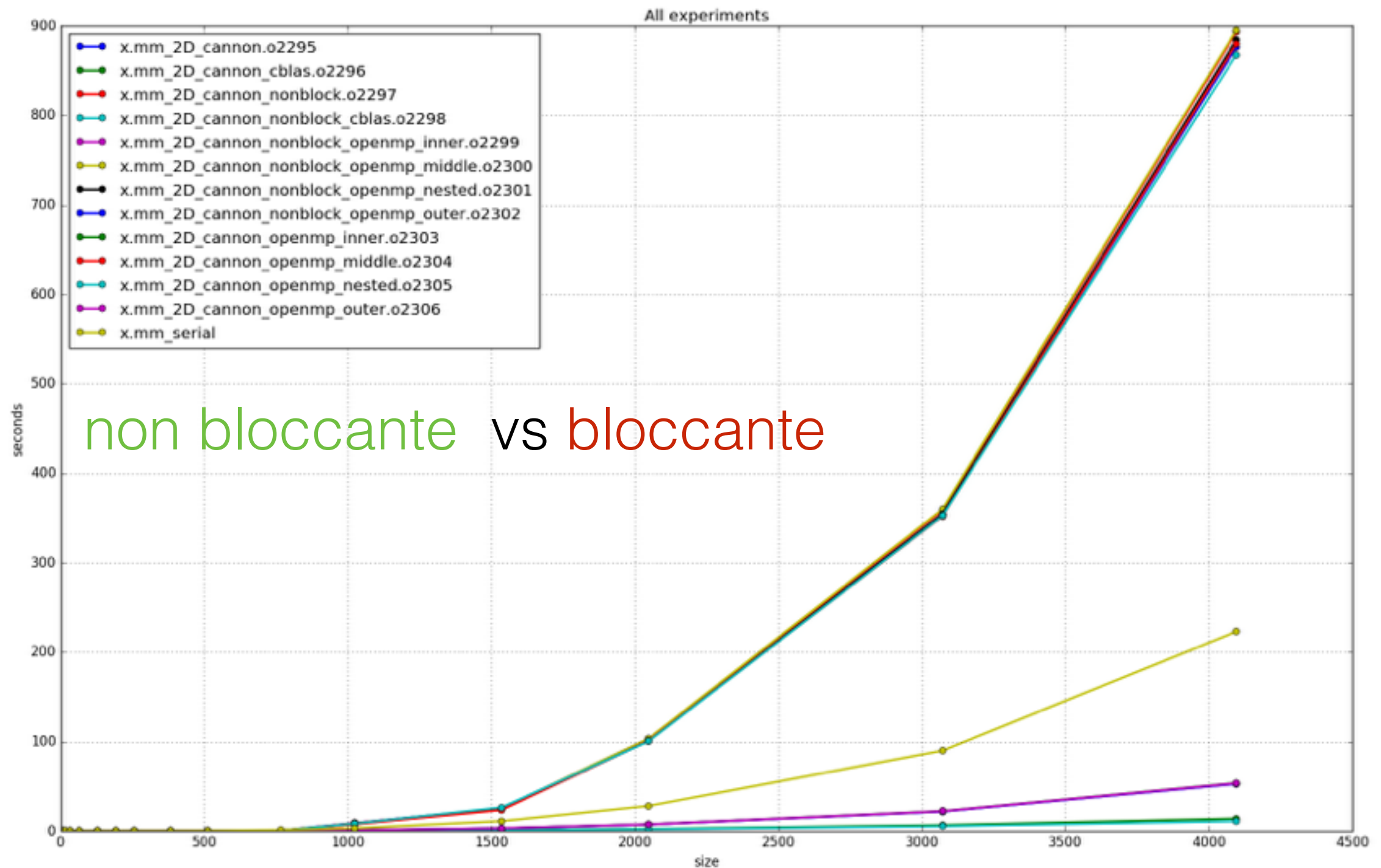
Risultato parallelo con OpenMP



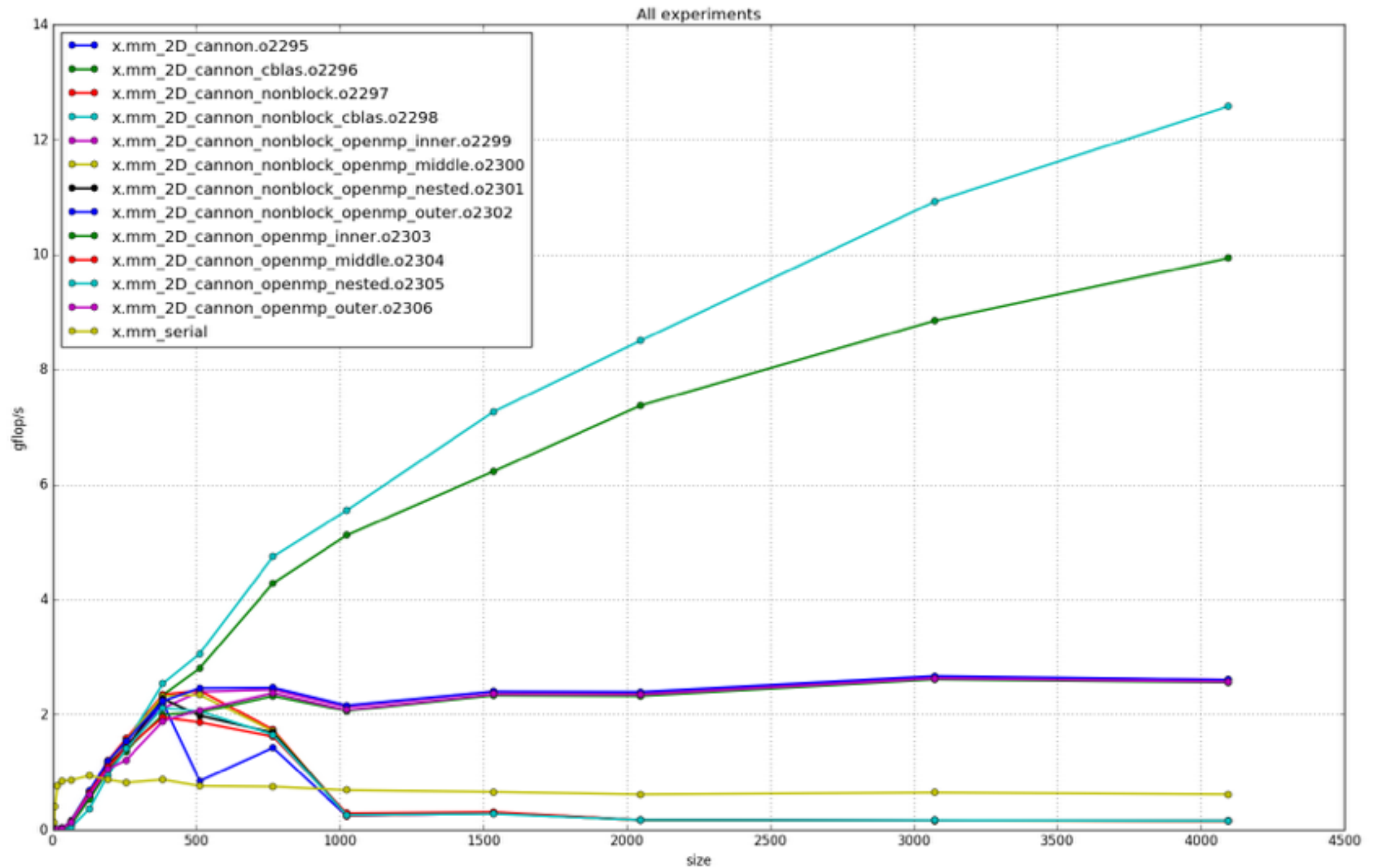
Risultato parallelo con CBLAS



All experiments: time



All experiments: gflop/s



Conclusioni

- Proof of concept, risultati interessanti
 - Ottimizzazione necessaria
- Funzionale:
 - I/O nell'applicazione
 - toggle ottimizzazioni runtime (1 binario)
- Performance:
 - OpenMP (più aggressivo)
 - OpenCL e GPU per moltiplicazioni locali