Multi-Threaded Composition of Weighted Finite-State Transducers

Bryan Jurish
Berlin-Brandenburg Academy of Sciences
jurish@bbaw.de

Kay-Michael Würzner
University of Potsdam
wuerzner@uni-potsdam.de

WATA 2012

Dresden, 29th May, 2012

Overview

The Big Idea

- The Situation
- The Approach

Parallel Composition Algorithms

- Master-Slave
- Peer-to-Peer

Experiments

- Materials
- Method
- Results

Concluding Remarks

— The Big Idea —

The Situation

No Free Lunch (anymore)

- CPU frequency growth stagnating
- Multiprocessor systems increasingly popular

$\rightsquigarrow$ *"horizontal" scaling* / multi-threading

(W)FST Composition

$$\mathcal{T}_3 = (\mathcal{T}_1 \circ \mathcal{T}_2)$$

- **Online:** lexical lookup, Viterbi decoding, parsing, ...
- **Offline:** lexicon compilation, statistical modelling, ...
- *no generic parallel implementation* (that we know of)

Amdahl's Law

$$S(N) = \frac{1}{(1-P)+\frac{P}{N}}$$

- Not all algorithms scale well horizontally ($P \ll 1$)
- For WFSTs, P may depend on *FST topology*
  $\rightsquigarrow$ *not all WFST compositions scale horizontally!*

The Basics

Definition

Given two ε-free WFSTs $\mathcal{T}_1 = \langle \Sigma, \Gamma, Q_1, q_{0_1}, F_1, E_1, \rho_1 \rangle$ and $\mathcal{T}_2 = \langle \Gamma, \Delta, Q_2, q_{0_2}, F_2, E_2, \rho_2 \rangle$, $\mathcal{T}_3 = (\mathcal{T}_1 \circ \mathcal{T}_2)$ is itself a WFST, with:

$$
\begin{aligned}
[\![\mathcal{T}_3]\!] &: (x, y) \mapsto \bigoplus_{z \in \Gamma^*} [\![\mathcal{T}_1]\!](x, z) \otimes [\![\mathcal{T}_2]\!](z, y) \\
\mathcal{T}_3 &= \langle \Sigma, \Delta, Q_1 \times Q_2, (q_{0_1}, q_{0_2}), F_1 \times F_2, E_3, \rho_1 \otimes \rho_2 \rangle \\
E_3 &= \biguplus_{\substack{(q_1, r_1, a, b, w_1) \in E_1, \\ (q_2, r_2, b, c, w_2) \in E_2}} \Big\{ \big((q_1, q_2), (r_1, r_2), a, c, w_1 \otimes w_2 \big) \Big\}
\end{aligned}
$$

Properties

- simple construction requires ε-free WFSTs
- commutative & complete semiring $\langle \mathbb{K}, \oplus, \otimes, \bar{1}, \bar{0} \rangle$
- worst-case $\mathcal{O}_{\text{time}} = \mathcal{O}(|E_1 \times E_2|)$

Serial Algorithm

$\texttt{compose}(\mathcal{T}_1 = \langle \Sigma, \Gamma, Q_1, q_{0_1}, F_1, E_1, \rho_1 \rangle, \mathcal{T}_2 = \langle \Gamma, \Delta, Q_2, q_{0_2}, F_2, E_2, \rho_2 \rangle)$

1 $Q \leftarrow \{(q_{0_1}, q_{0_2})\}$ /* initialize */

2 $V \leftarrow \{(q_{0_1}, q_{0_2})\}$ /* visitation queue */

3 **while** $V \neq \emptyset$ **do**

4 $(q_1, q_2) \leftarrow \texttt{pop}(V)$ /* visit state */

5 **if** $(q_1, q_2) \in F_1 \times F_2$ **then** /* final state */

6 $F \leftarrow F \cup \{(q_1, q_2)\}$

7 $\rho(q_1, q_2) \leftarrow \rho_1(q_1) \otimes \rho_2(q_2)$

8 **foreach** $(e_1, e_2) \in E[q_1] \times E[q_2]$ **with** $\mathrm{o}[e_1] = \mathrm{i}[e_2]$ **do** /* align edges */

9 **if** $(\mathrm{n}[e_1], \mathrm{n}[e_2]) \notin Q$ **then**

10 $Q \leftarrow Q \cup \{(\mathrm{n}[e_1], \mathrm{n}[e_2])\}$

11 $V \leftarrow V \cup \{(\mathrm{n}[e_1], \mathrm{n}[e_2])\}$ /* enqueue for visitation */

12 $E \leftarrow E \cup \{(q_1, q_2), (\mathrm{n}[e_1], \mathrm{n}[e_2]), \mathrm{i}[e_1], \mathrm{o}[e_2], \mathrm{w}[e_1] \otimes \mathrm{w}[e_2]\}$

13 **return** $\mathcal{T}_3 = \langle \Sigma, \Delta, Q, (q_{0_1}, q_{0_2}), F, E, \rho \rangle$

The Approach

Parallel State Visitation (lines 4–12)

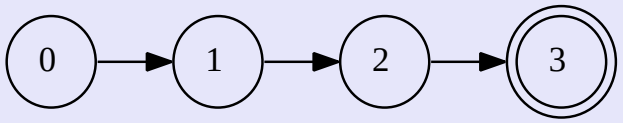
- breadth-first search of output states (V : FIFO)
- distributed output data (Q, F, E, ρ)
- shared visitation queue (V)

Amdahl's Law Revisited

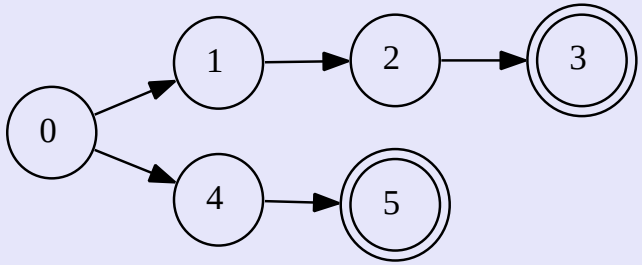
$$S_{\max} :\approx \frac{|Q|}{1+\mathrm{depth}(\mathcal{T}_3)} = \frac{|Q|}{1+\max_{q\in Q}\min_{\pi\in\Pi(q_0,q)}|\pi|}$$

$$P = 1 - \frac{1}{S_{\max}}$$

- assumes constant (average) state complexity
- worst-case breadth-first visitation



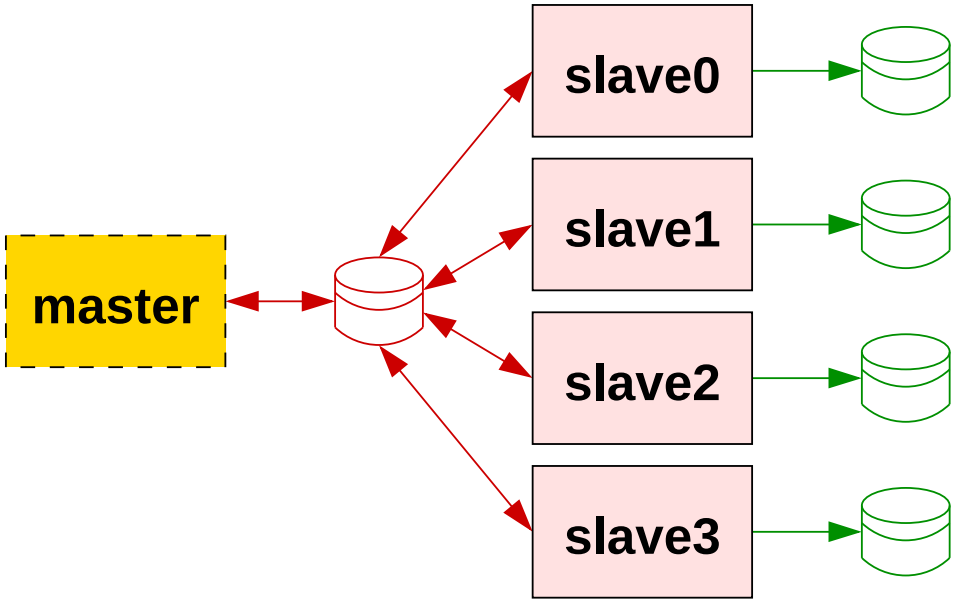
$S_{\max} = 1$; $P = 0$



$S_{\max} = \frac{3}{2}$; $P = \frac{1}{3}$

— Algorithms —

Algorithm (Sketch): Master-Slave



Superordinate Distribution of Work

- state-pairs (q_1, q_2) passed to slaves for visitation

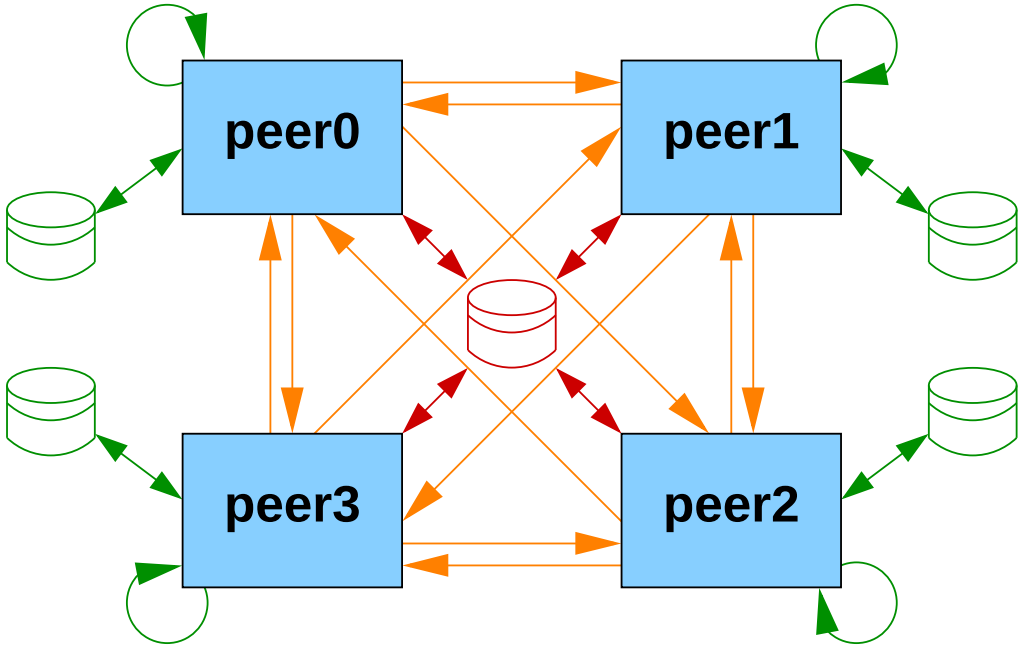
Slave Tasks

- align & expand transitions, globally enqueue visitation requests

Shared Global Data

- V : visitation queue — $V \subseteq Q_1 \times Q_2$
- Q : visited states — $Q \subseteq Q_1 \times Q_2$
- n_q : output state counter — *(for serialization)*
- n_up: number of tasks currently assigned — *(for termination)*

Algorithm (Sketch): Peer-to-Peer



State Partitioning Function

- peer i visits states with $r(q_1, q_2) = i$

$$r : (q_1, q_2) \mapsto \left\lfloor \frac{q_1 + q_2}{2} \right\rfloor \bmod N$$

Peer-to-Peer Message Passing

- messages are aligned transitions (e_1, e_2)
- sender: $r(\mathrm{p}[e_1], \mathrm{p}[e_2]) \rightsquigarrow$ receiver: $r(\mathrm{n}[e_1], \mathrm{n}[e_2])$

$$V \in \wp(E_1 \times E_2)^{N \times N}$$

Shared Global Data

- n_q : output state counter *(for serialization)*
- n_up: number of messages currently enqueued *(for termination)*

— Experiments —

Experiments

Materials

- 2,266 randomly generated WFSTs $\mathcal{T}$
  - trie spine + random arcs — $\mathrm{depth}(\mathcal{T}) \leq 32$
  - (piecewise-) uniform sampling — $|Q_{\mathcal{T}}|, |E_{\mathcal{T}}|, |\Sigma|$
  - "embarrassingly parallel" topology — $P_{(\mathcal{T}^{-1} \circ \mathcal{T})} > 99\%$
- algorithms implemented in C++ — *g++ v4.4.5*
- hexadecacore test machine — *16 physical cores*

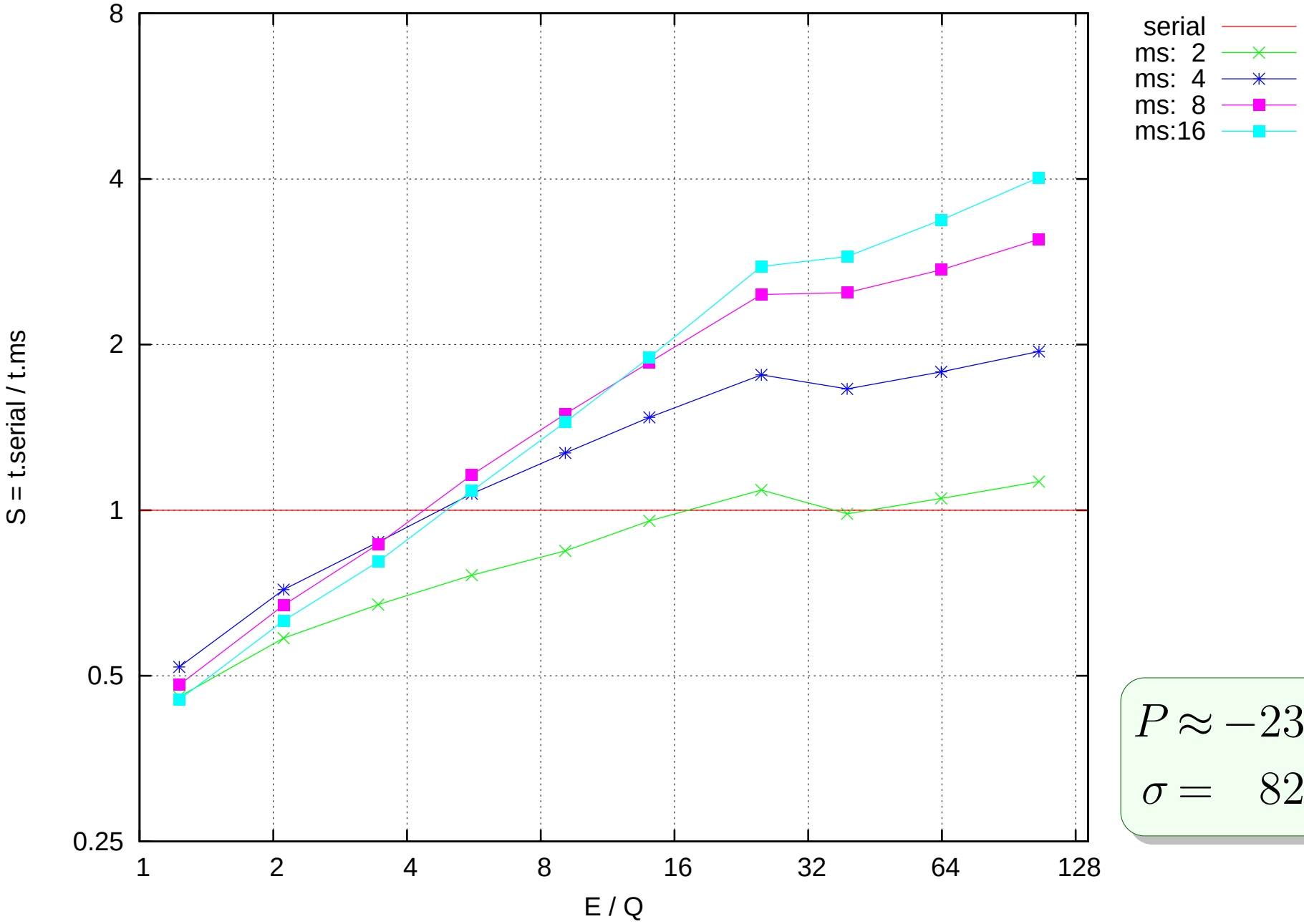
Method

- for each generated $\mathcal{T}$, compute $(\mathcal{T}^{-1} \circ \mathcal{T})$
  - sample selection filter — $\frac{1}{64}\ \mathrm{sec} \leq t_{\mathrm{serial}} \leq 8\ \mathrm{sec}$
  - varied number of threads — $N \in \{1, 2, 4, 8, 16\}$

Evaluation

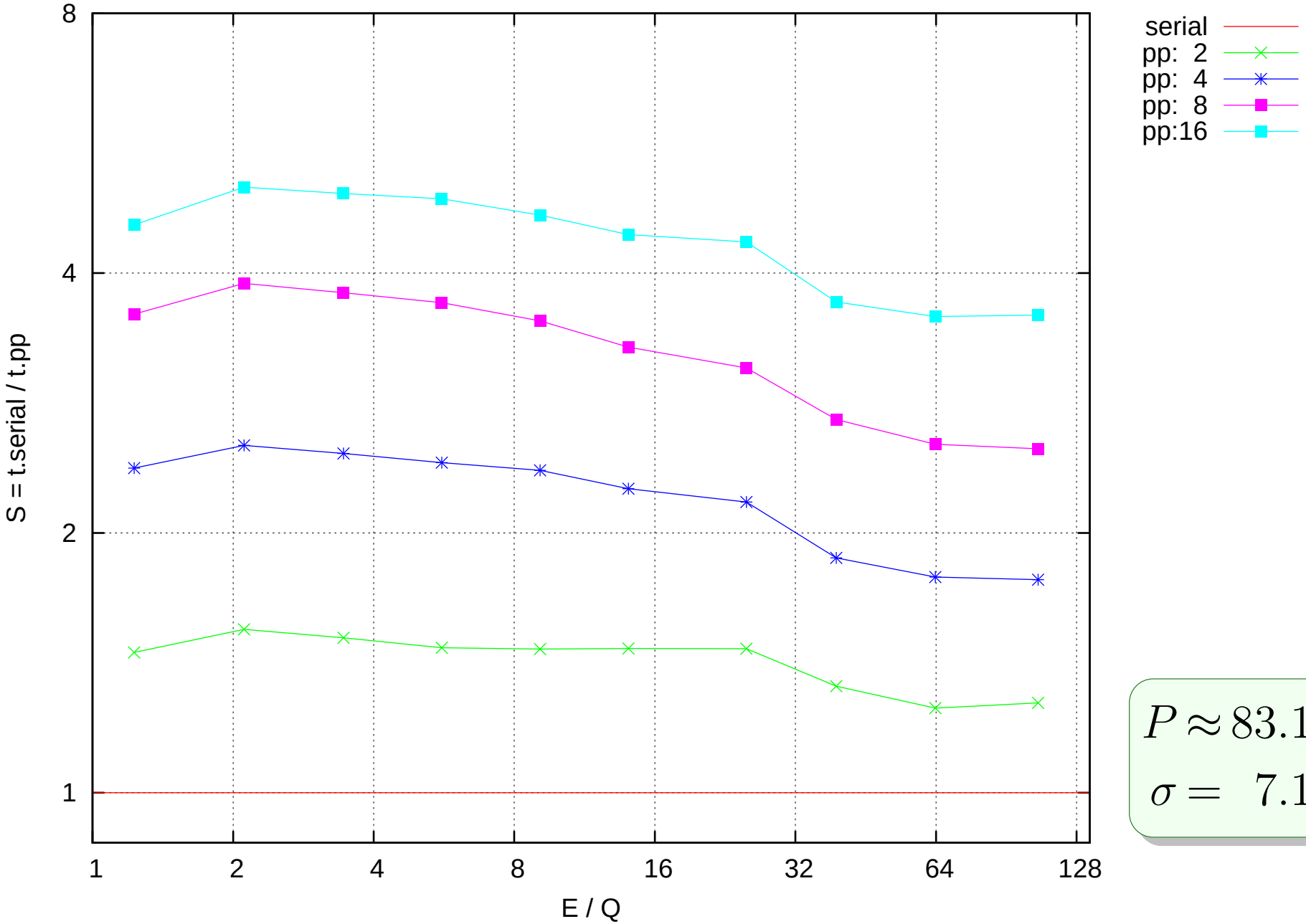
- average running time — *8 iterations per configuration*
- structural properties of $\mathcal{T}, (\mathcal{T}^{-1} \circ \mathcal{T})$ — $|Q|, |E|, \ldots$

Results: Master-Slave



$$P \approx -23.5\%$$
$$\sigma = \quad 82.3\%$$

Results: Peer-to-Peer



$$P \approx 83.1\ \%$$
$$\sigma = 7.18\%$$

Concluding Remarks

Summary

- No (more) Free Lunch
  - *parallelization of “traditional” serial algorithms*
- Amdahl’s Law Applied
  - *maximum speedup depends on FST topology*
- Sharing (data) Hurts
  - *distributed synchronization improves performance*

Future Directions

- improve sampling procedure
- reduce p2p communication overhead?
- massive parallel architectures (CUDA, OpenCL)?

The End

Thank you for listening!

— Addenda —

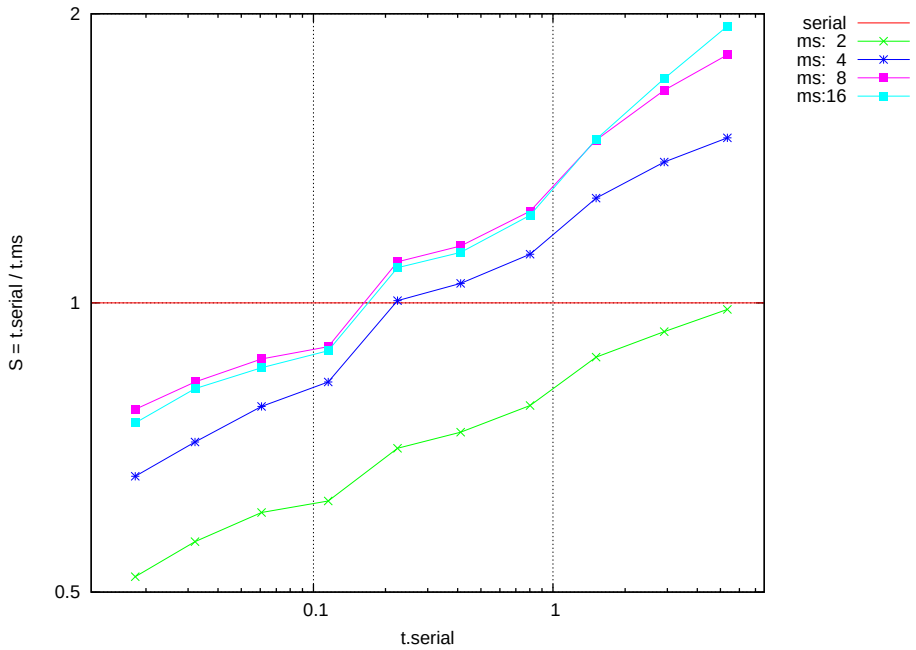
2d Plots

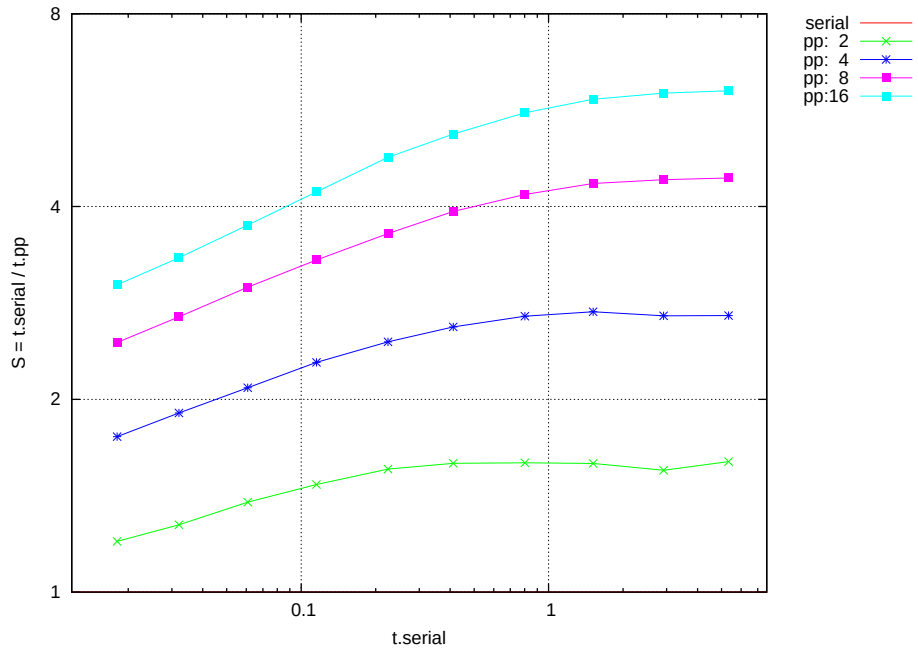
- $t_{\text{serial}} : S$
- $E : S$
- $E/Q : S$

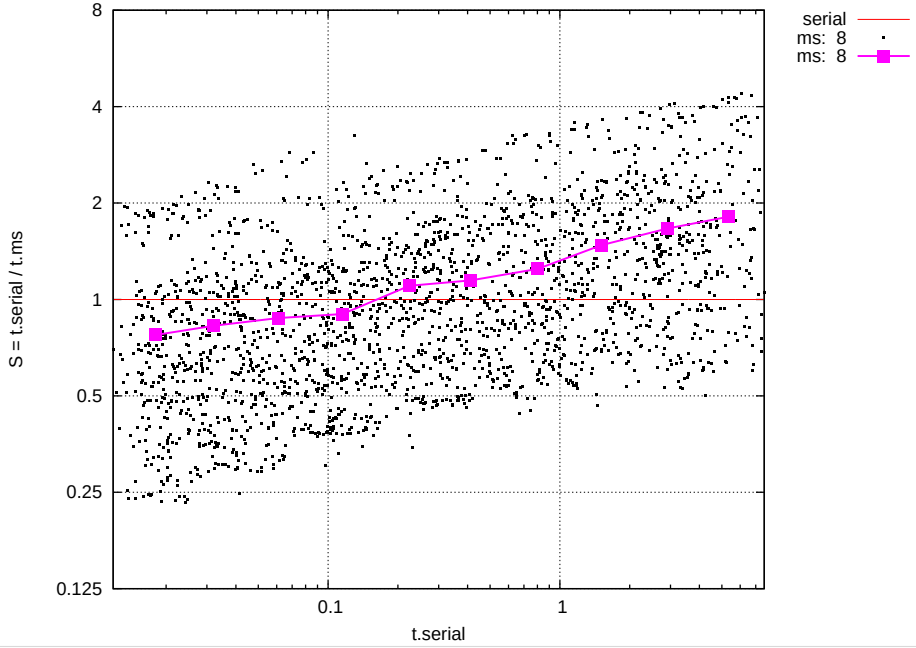
3d Plots

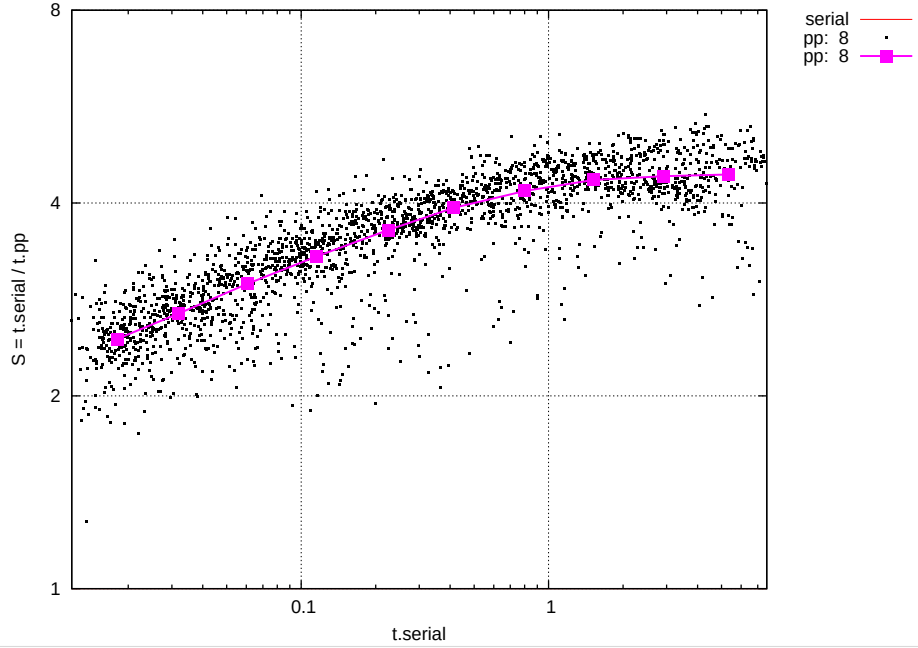
- $E/Q : N : S$
- $t_{\text{serial}} : N : S$
- $Q : E$: histogram
- $t_{\text{serial}} : E/Q$: histogram

Plots: 2d: $t_{\text{serial}} : S$

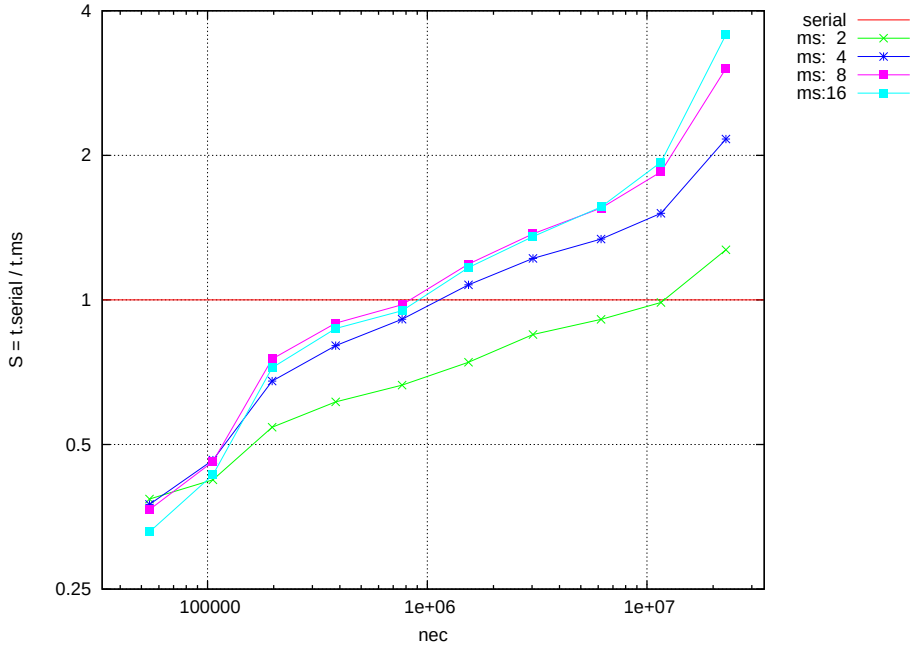


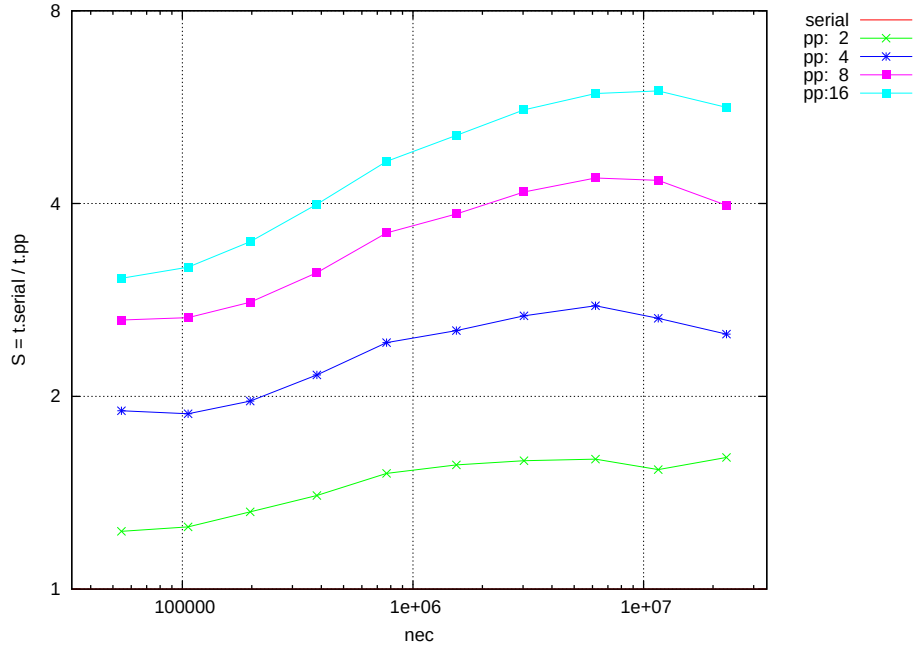


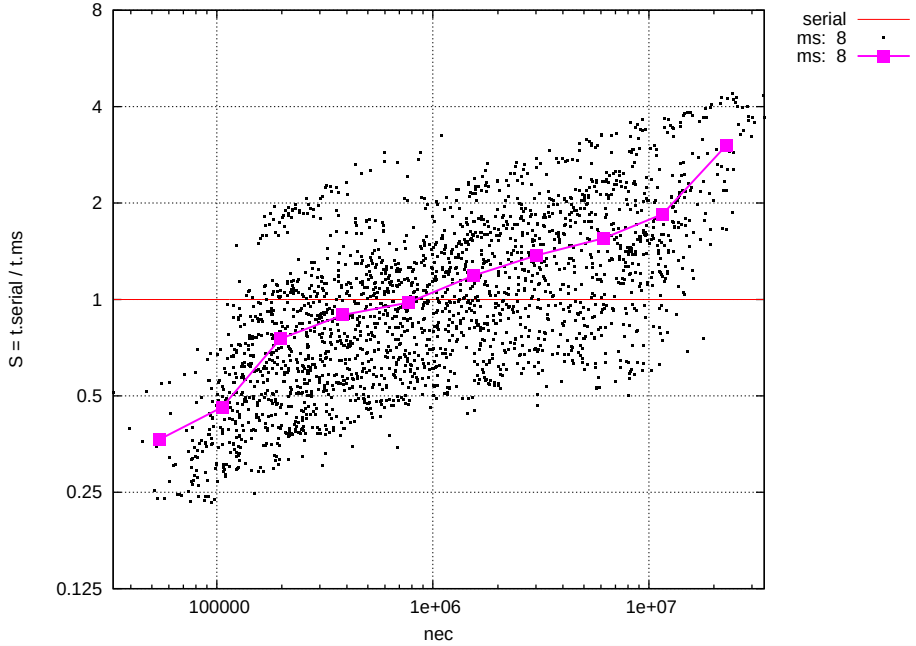


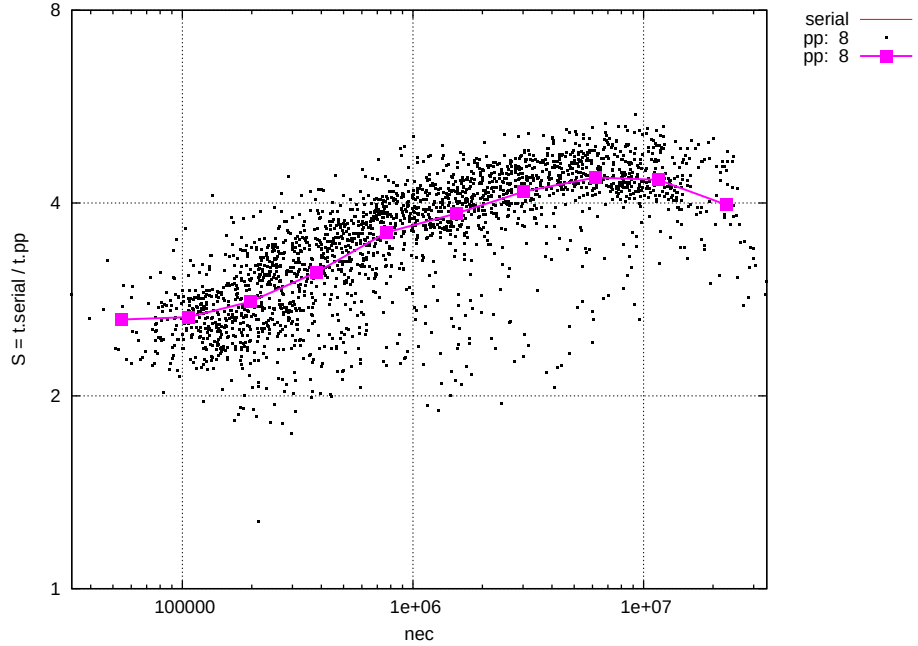


Plots: 2d: $E : S$

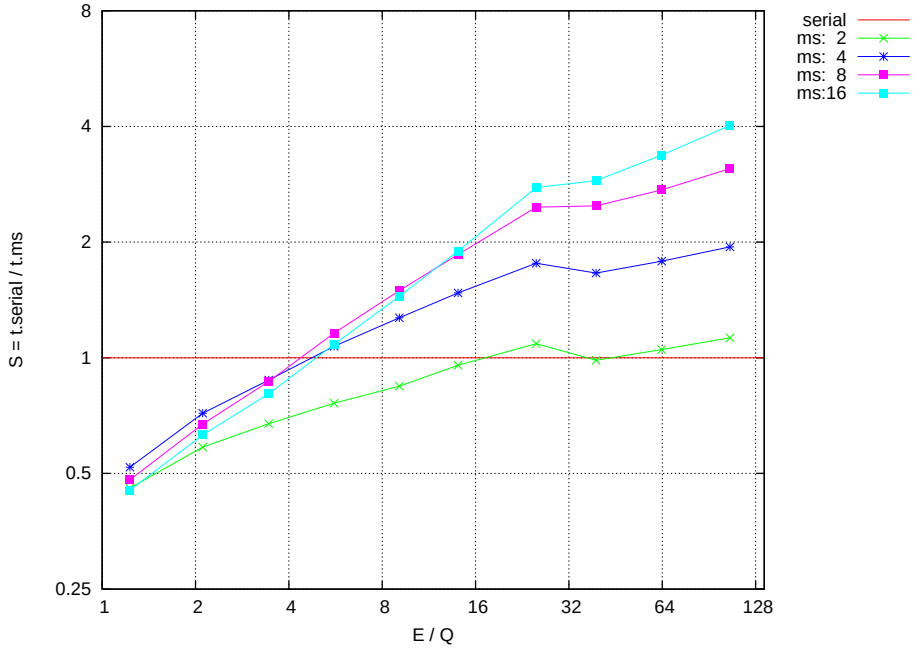

serial
ms: 2
ms: 4
ms: 8
ms:16
S = t.serial / t.ms
nec
4
2
1
0.5
0.25
100000
1e+06
1e+07

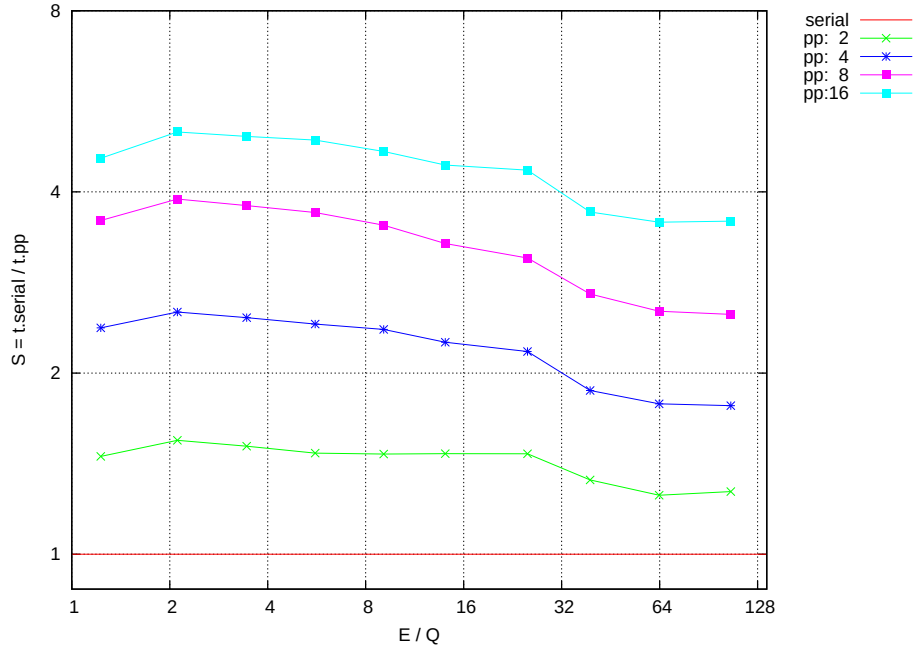

serial
pp: 2
pp: 4
pp: 8
pp:16
S = t.serial / t.pp
nec
8
4
2
1
100000
1e+06
1e+07

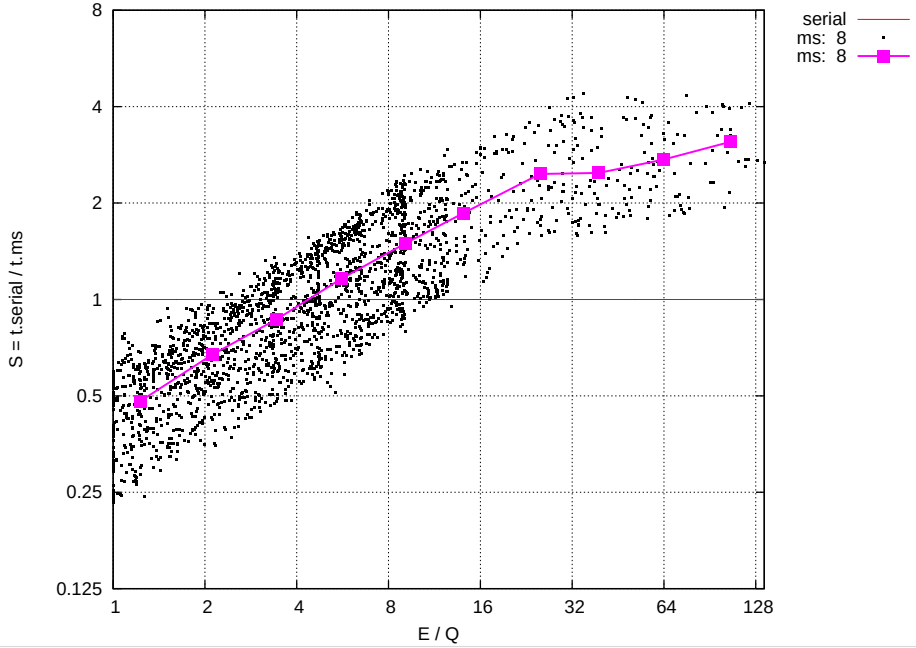

serial
ms: 8
ms: 8
S = t.serial / t.ms
nec
8
4
2
1
0.5
0.25
0.125
100000
1e+06
1e+07

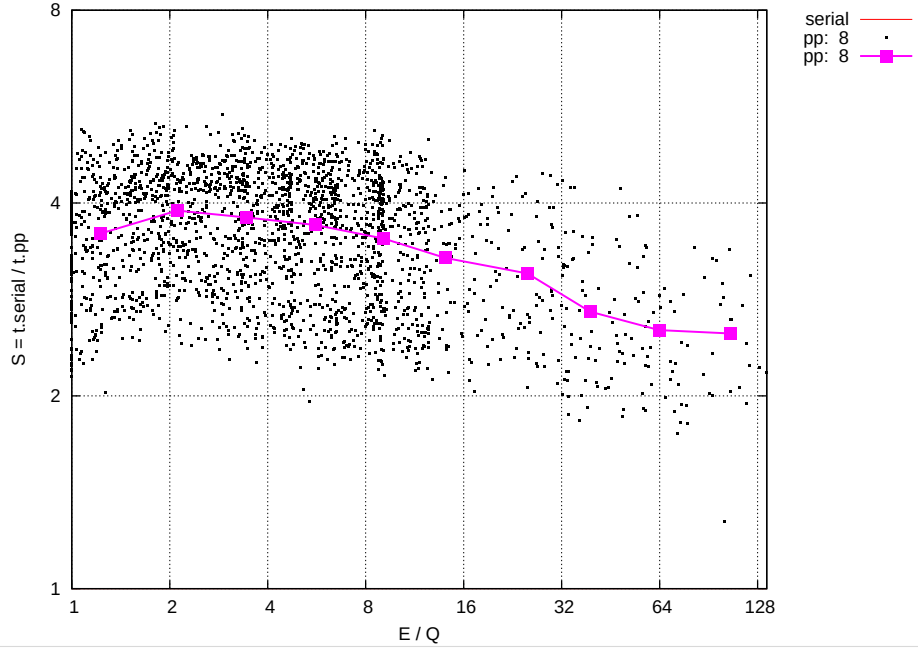

serial
pp: 8
pp: 8
S = t.serial / t.pp
nec
8
4
2
1
100000
1e+06
1e+07

Plots: 2d: $E/Q : S$


serial
ms: 2
ms: 4
ms: 8
ms:16
S = t.serial / t.ms
E / Q
8
4
2
1
0.5
0.25
1
2
4
8
16
32
64
128

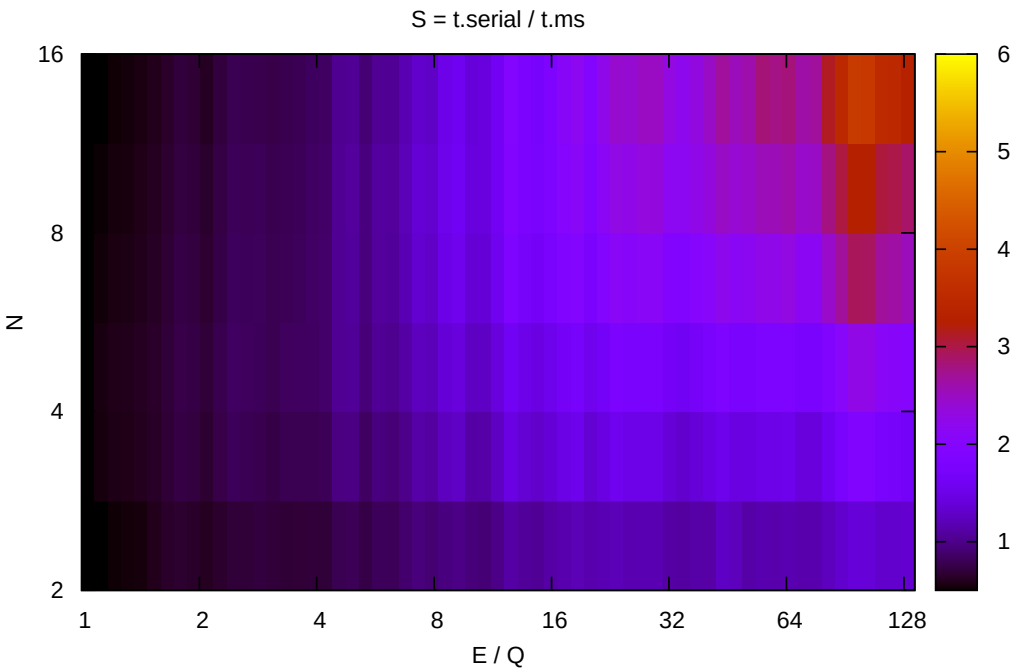

serial
pp: 2
pp: 4
pp: 8
pp:16
S = t.serial / t.pp
E / Q
8
4
2
1
1
2
4
8
16
32
64
128


serial
ms: 8
ms: 8
S = t.serial / t.ms
E / Q
8
4
2
1
0.5
0.25
0.125
1
2
4
8
16
32
64
128

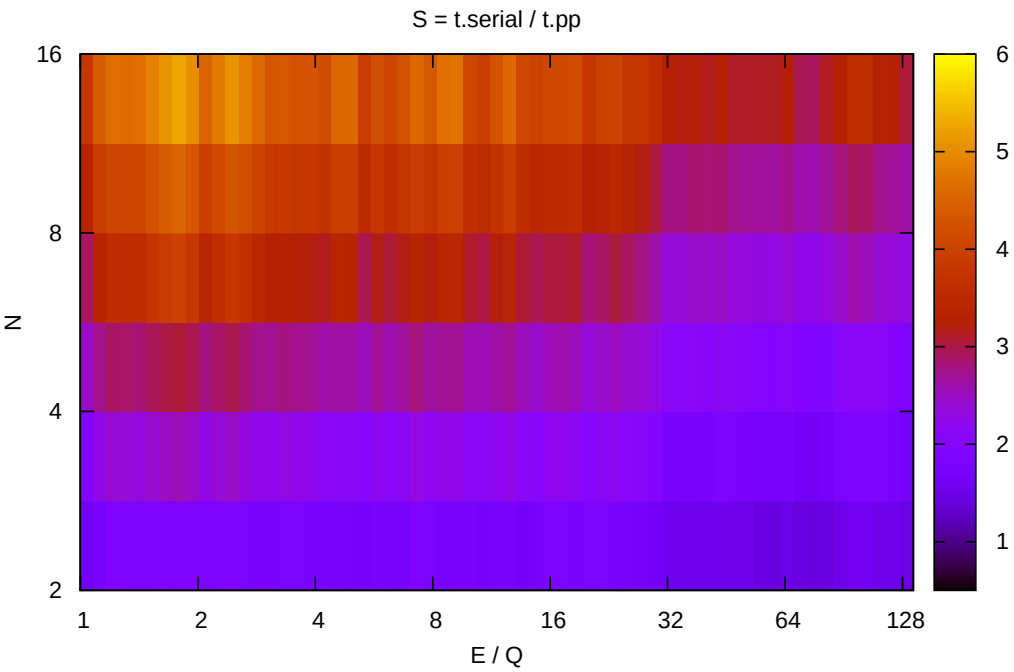

serial
pp: 8
pp: 8
S = t.serial / t.pp
E / Q
8
4
2
1
1
2
4
8
16
32
64
128

Plots: 3d: $E/Q : N : S$

master-slave

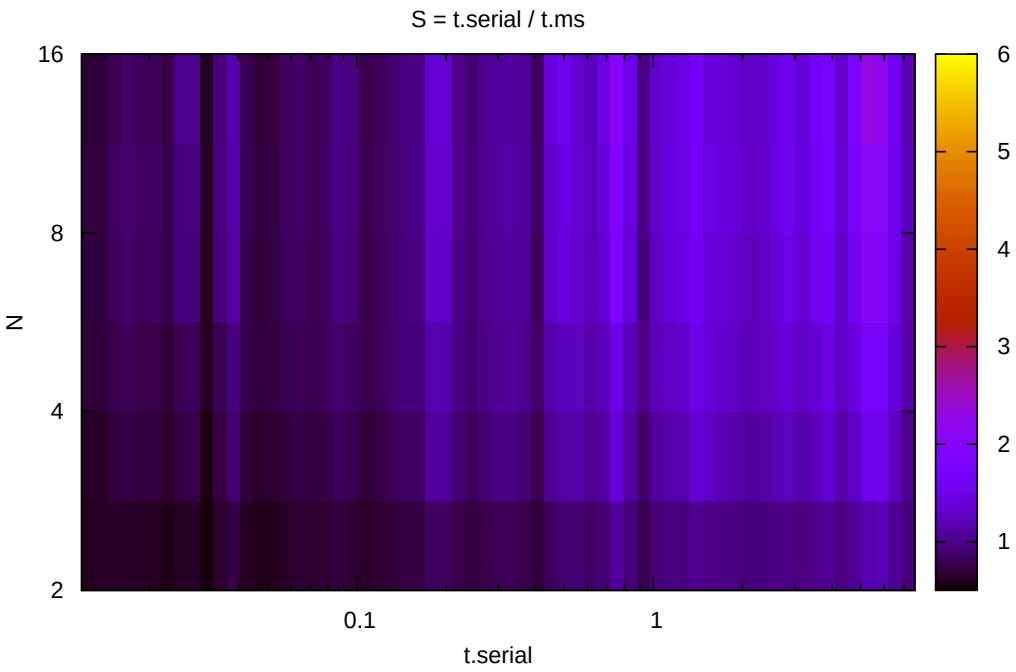


peer-to-peer

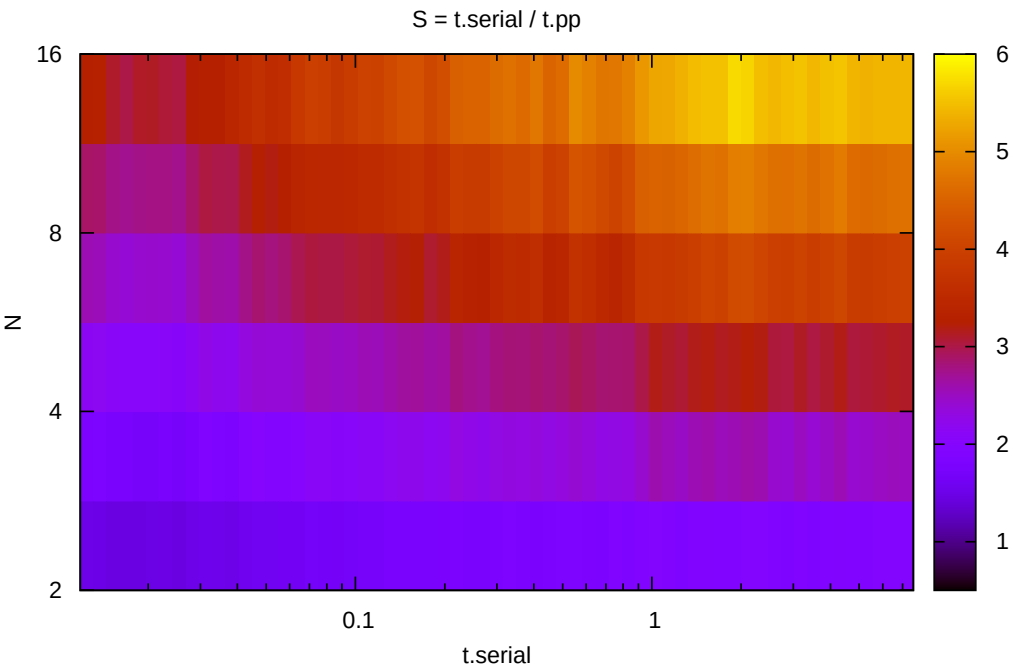


Plots: 3d: $t_{\text{serial}} : N : S$

master-slave

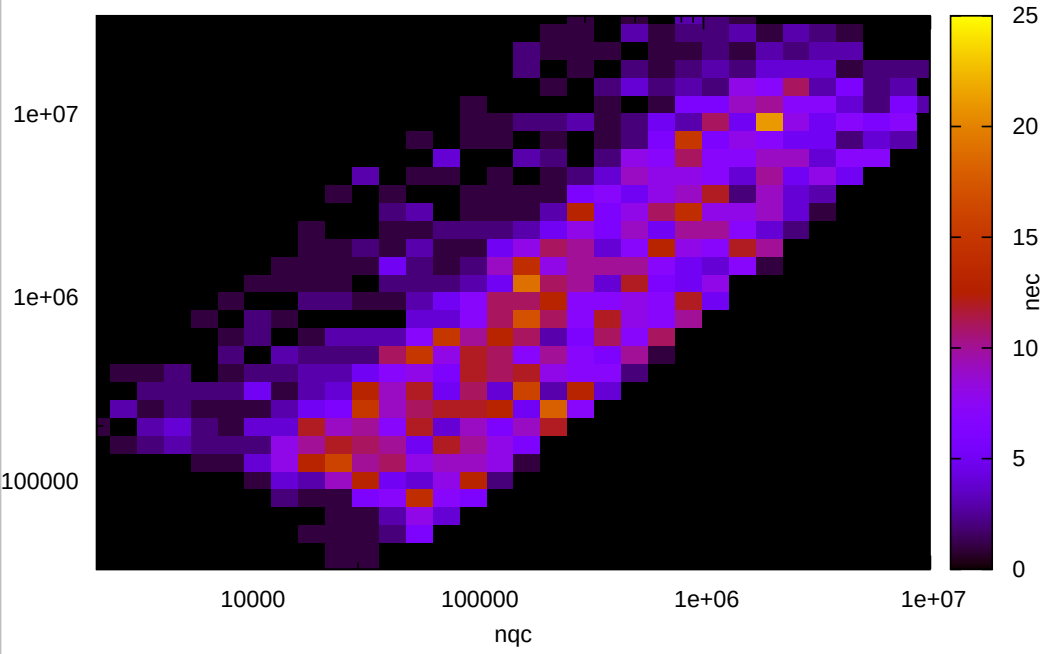


peer-to-peer

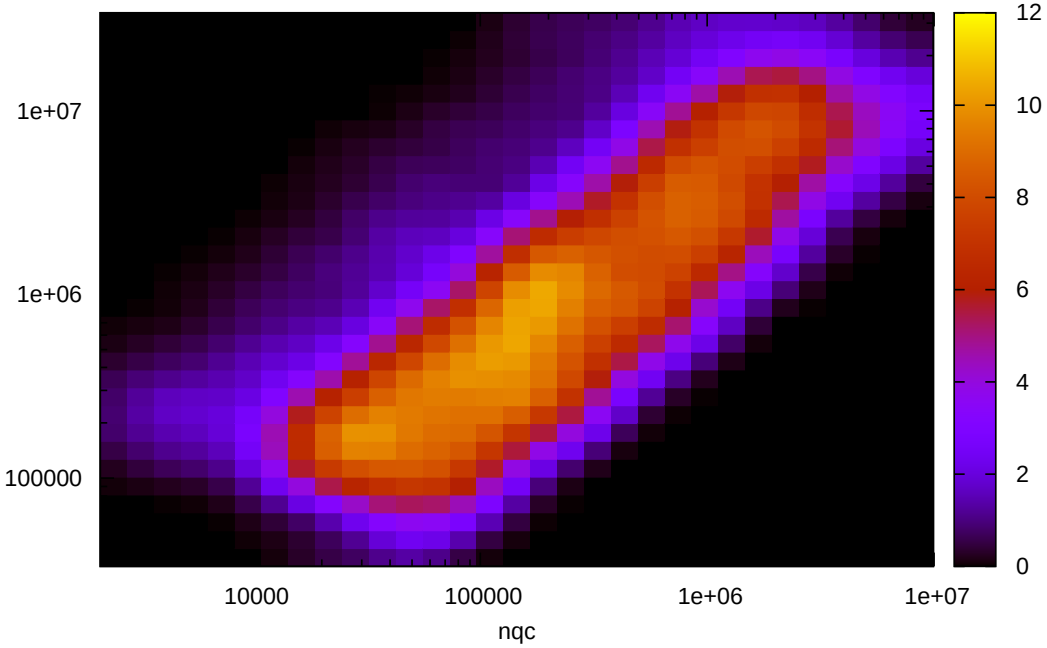


Plots: 3d: $Q : E$: histogram

raw

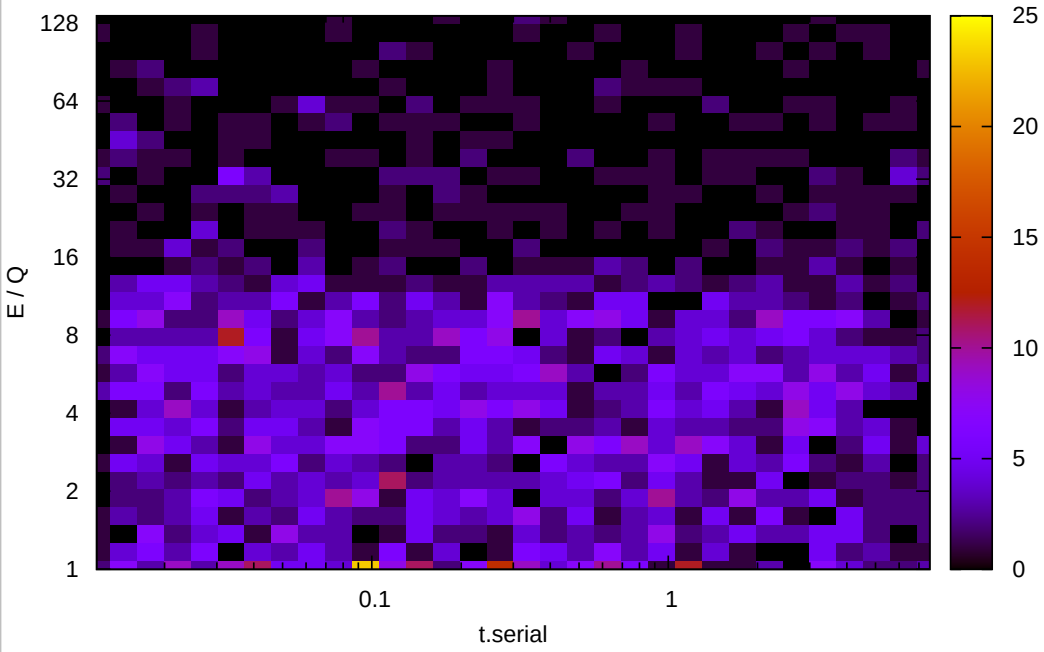


smoothed



Plots: 3d: $t_{\text{serial}} : E/Q$: histogram

raw



smoothed