ADVENTURES IN MICRO-BENCHMARKING

Yossi Gil, Keren Lenz, Yuval Shimron
The Technion

WHY IS MICRO-BENCHMARKING INTERESTING

Because, deep inside, we still believe that the whole is the sum of its parts:

What would be the impact of replacing a component ***A*** by component ***B*** in a program ***P***?

Compare $\boldsymbol{P_A}$ with $\boldsymbol{P_B}$?

Why not compare ***A*** and ***B*?** The results would be then valid for <u>all</u> **P***!*

WHY IS MICRO-BENCHMARKING DIFFICULT?

- The CPU / memory model / operating system /JIT / virtual machine / … stack is very complicated
  - No one fully understands it all.
- Microbenchmarking can never be done directly.
  - A java method may consume less than 10 ns of CPU time!
- Exponential blowup in the parameters space:
  - CPU model / # cores / Memory Size
  - Softwae environment: O/S, etc.
  - JVM
  - JVM parameters
- JVM pecularities:
  - Multi-threading (13 threads in a “thread free” program)
  - Garbage collection
  - JIT optimization

BACKGROUND I
ANONYMOUS ECOOP PAPER

- **Berlin 2007**: Timing results (hidden in tables and in slides):
  - *X*: time to process a data structure of size *n*
  - *Y*: time to process a data structure of size *10n*
  - *But* $X < Y/10$
- **Berlin 2007 (lunch break): "***I can show you the results right here, check it out on my laptop…"*
- **Berlin 2007 (after lunch break): "***I will email your the results…"*
- **Portland 2011:** …

BACKGROUND II
SPACE OPTIMIZATION OF JAVA'S HASH MAP

- **Anonymous Reviewer**: *"I do not believe your timing results… Did you acount for garbage collection cycles? Did you allow sufficient time for the JIT to become active? Did you experiment with different VM flags?"*
- **Author response: "***I am confident that the results are correct!"*
- **Rejection Aftermath: "***There must be a system in this madness!"*
- **Well known authority: "***Billions and billions of runs!"*

BACKGROUND III: **THE LAW OF LARGE NUMBERS**

$$\Pr\left(\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}x_n=\mu\right)=1$$

- Let $x_1,\cdots,x_n$ be a sequence of measurements of some pheonmenon (a random variable). Then, with probability 1, the average of the sequence converges to the expected value of the phenomenon.
- Sounds trivial... But, very powerful.

<u>No matter what</u>, if you repeat your mesaurement sufficiently many times, and then average over all measurements, you will get the “right” result.

BACKGROUND IV:
THE CENTRAL LIMIT THEOREM

$$\lim_{n\to\infty} \Pr\left[\sqrt{n}\left(\frac{1}{n}\sum_{i=1}^{n} x_n - \mu\right) \le z\right] = \Phi\left(\frac{z}{\sigma}\right)$$

- Let $x_1, \cdots, x_n$ be a sequence of measurements of some pheonmenon .
- Then, the average of the sequence, the deviation from the expected value follows a ***normal distribtution***.
- Further, the standard deviation of this distribution is invesely proportional to the square root of the sequence length.

SO, WHY SHOULD ANYTHING GO WRONG?

<u>`REPEAT`</u>

- Experiment:
  A. Define a simple microbenchmark
  B. Neutralize all “noise” factors: GC, JIT, background processors, Bill Gates, etc.
  C. Carefully read Josh Bloch’s warnings
  D. Use billions and billions of runs.
- Plot/Tabulate/Blah Blah the results

<u>`UNTIL`</u> “happiness achieved”;

OUR WAY OUT OF THIS INFINITE LOOP
REPORT DIFFICULTIES ENCOUNTERED

- **Instability of the Virtual Machine**
  - Different invocations give statistically different results
  - Disabling the JIT makes things worse
- **The steady state rules:**
  I. The “steady state” will be different in different invocations
  II. The “steady state” may suddenly change during the same invocation
  III. At any given time during an invocation may have more than one steady state
- **Long Memory of the Virtual Machine:**
  - Even short execution may contaminate the JIT
  - So, what’s the point in microbenchmarking?

EXPERIMENTAL SETTING

- **Hardware** * : Intel Core 2 Quad CPUQ9400, 8GB
- **Java**: OpenJDK, IcedTea6 1.9.8,JVM 1.6.0_20
- **O/S:** Ubuntu * 10.04.2 *
- **Run mode:** single user (`telinit 1`), text mode (no GUI), clean boot, no network, batch execution,wait for small uptime before starting
- **Benchmarked code * :** function `get` in the JRE's standard collection class `HashMap`
  - Bit operations: rotate/XOR
  - Memory dereferencing
  - Comparisons
  - Conditionals/Iterations/function call / return
  - No dynamic dispatch
  - No memory allocation/dispatch

BENCHAMRKING PROCEDURE

- Minimal * pre-processing in the main Java program.
- Monitor each measurement using MX beans found in class `ManagementFactory`
- Discard measurement if:
  - GC cycle detected.
  - JIT cycle detected
  - Load/Unload event detected
- Function `getCaller()` calls `get()` *n* times, *n* = 1,152
- Call function `getCaller()` *m* times, *m* ~ 6,000 (total runtime is 100ms *).
- Measure the time of each call
  - Note this runtime `getCaller()` is expected to follow a normal distribution.
- Repeat *r* times, *r* = 20,000

RAW RESULTS OF SEVEN INVOCATIONS

No	Mean	SD	Median	MAD	MIN	MAX
1	15.27	0.0802	15.27	0.0509	14.95	15.69
2	15.17	0.0768	15.17	0.0454	14.74	15.69
3	15.27	0.0726	15.24	0.0451	14.93	15.67
4	15.38	0.0727	15.38	0.0430	15.06	15.81
5	15.29	0.0867	15.30	0.0495	14.95	15.68
6	15.10	0.0722	15.11	0.0420	14.83	15.51
7	15.24	0.0929	15.25	0.0559	14.87	15.60

Looks decent, right?

NO! SOMETHING IS OBVIOUSLY WRONG!!!

Largest Average

No	Mean	SD	Median	MAD	MIN	MAX
1	15.27	0.0802	15.27	0.0509	14.95	15.69
2	15.17	0.0768	15.17	0.0454	14.74	15.69
3	15.27	0.0726	15.24	0.0451	14.93	15.67
4	15.38	0.0727	15.38	0.0430	15.06	15.81
5	15.29	0.0867	15.30	0.0495	14.95	15.68
6	15.10	0.0722	15.11	0.0420	14.83	15.51
7	15.24	0.0929	15.25	0.0559	14.87	15.60

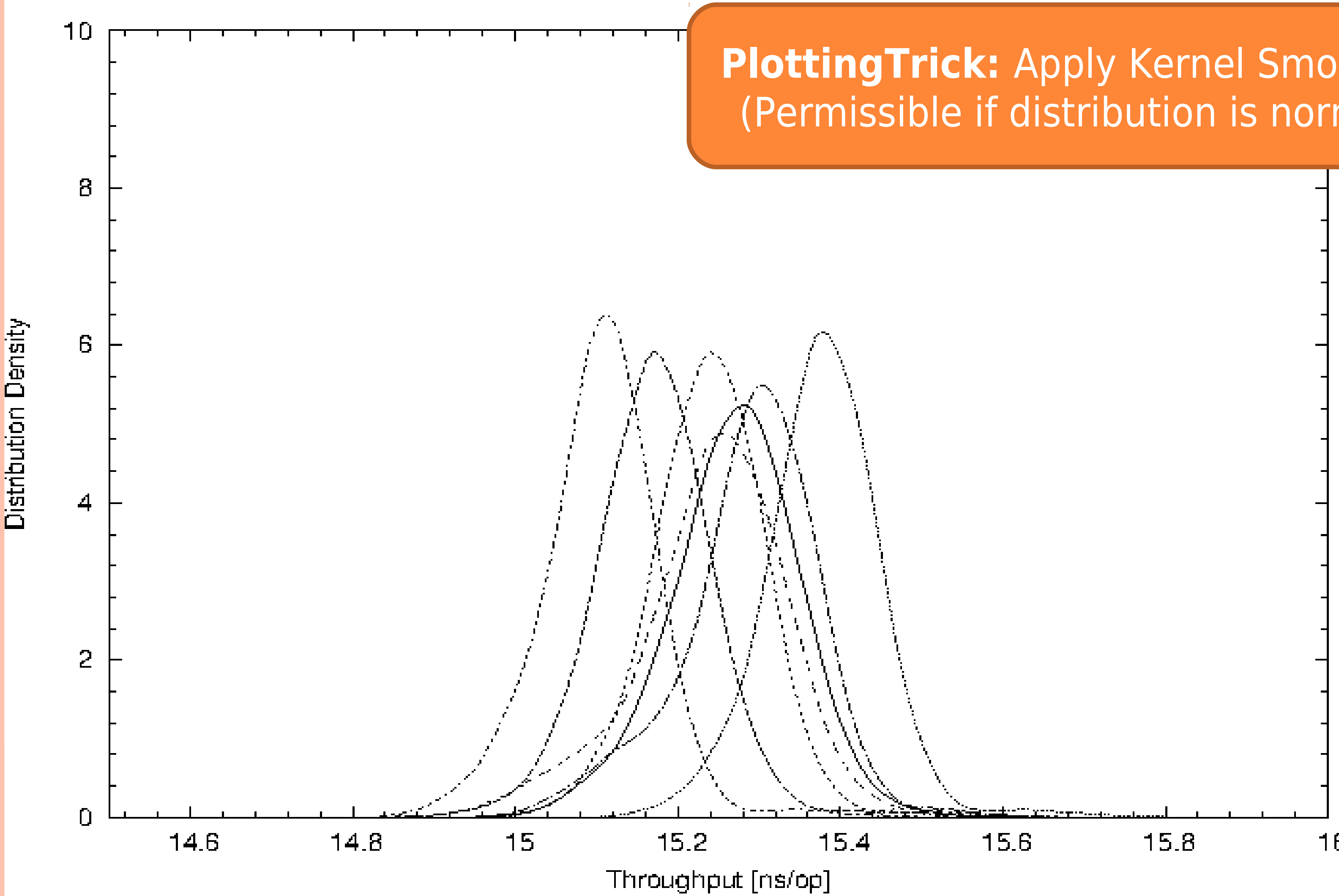
Smallest Average

Largest SD

Difference is 3 Standard Deviations!!!

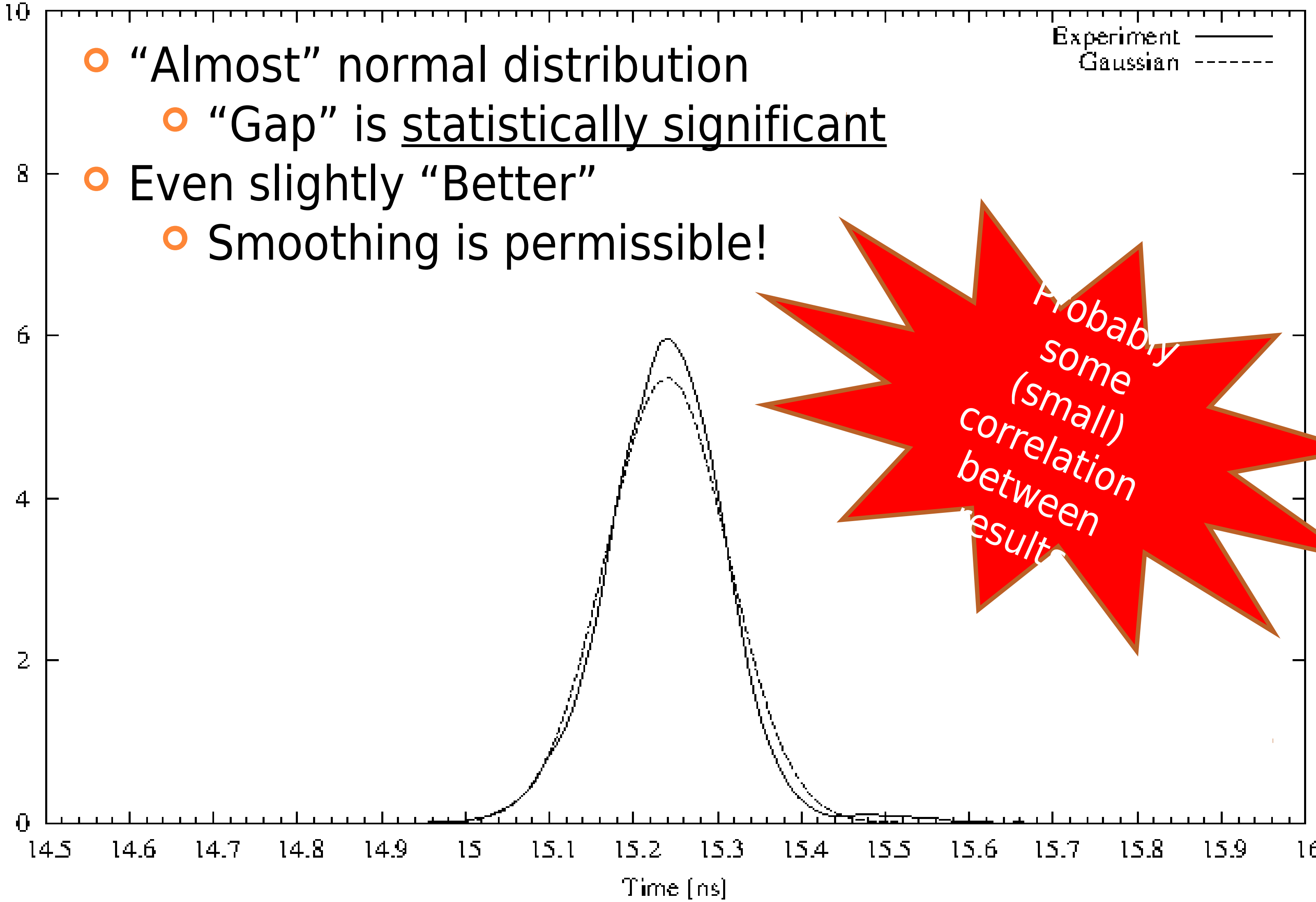
DISTINCT INVOCATIONS DO NOT SHARE A SINGLE STEADY STATE!

PlottingTrick: Apply Kernel Smoothing (Permissible if distribution is normal)



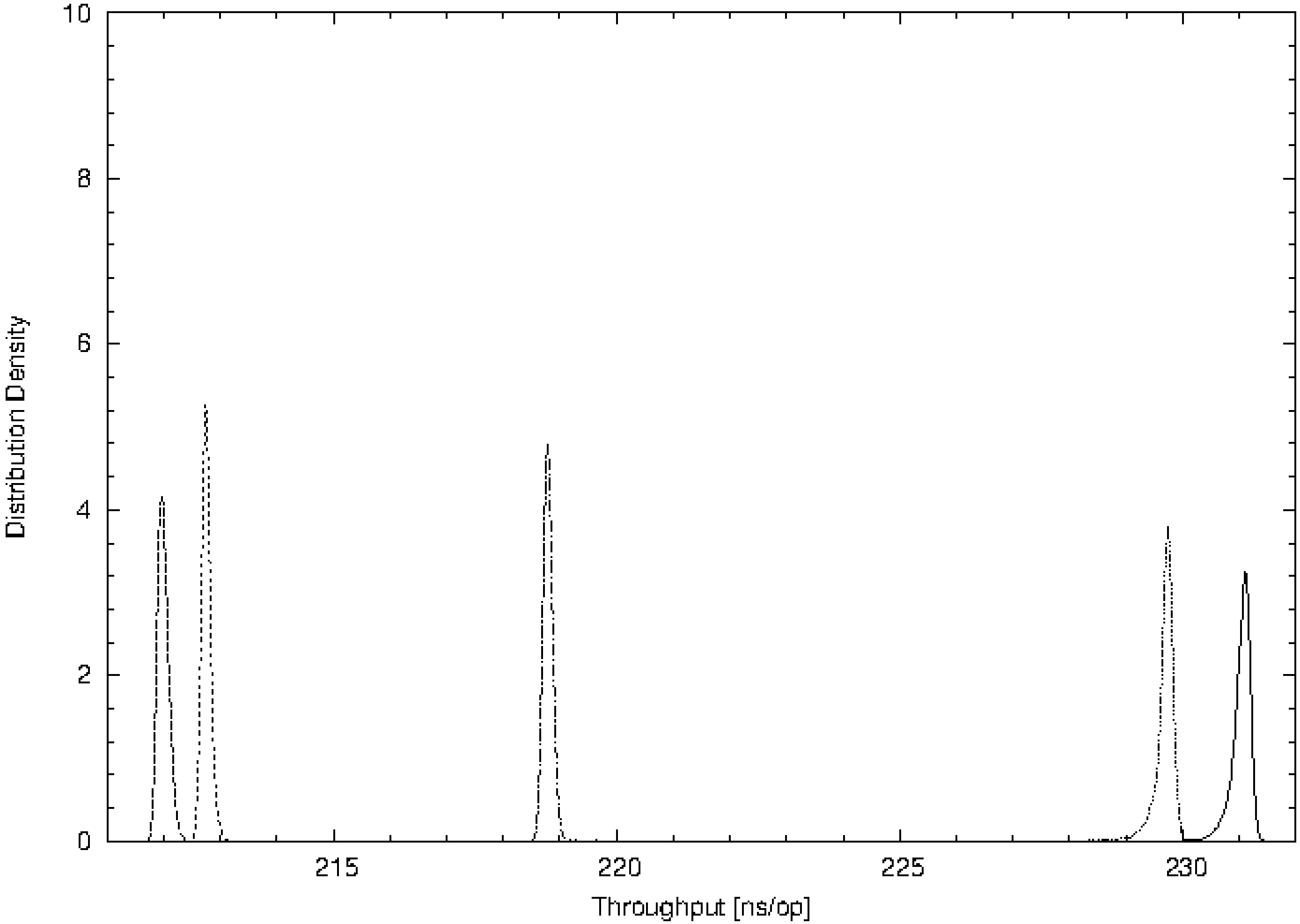
ARE THE MEASUREMENT RESULTS IN A SINGLE INVOCATION NORMALLY DISTRIBUTED?

- “Almost” normal distribution
  - “Gap” is <u>statistically significant</u>
- Even slightly “Better”
  - Smoothing is permissible!

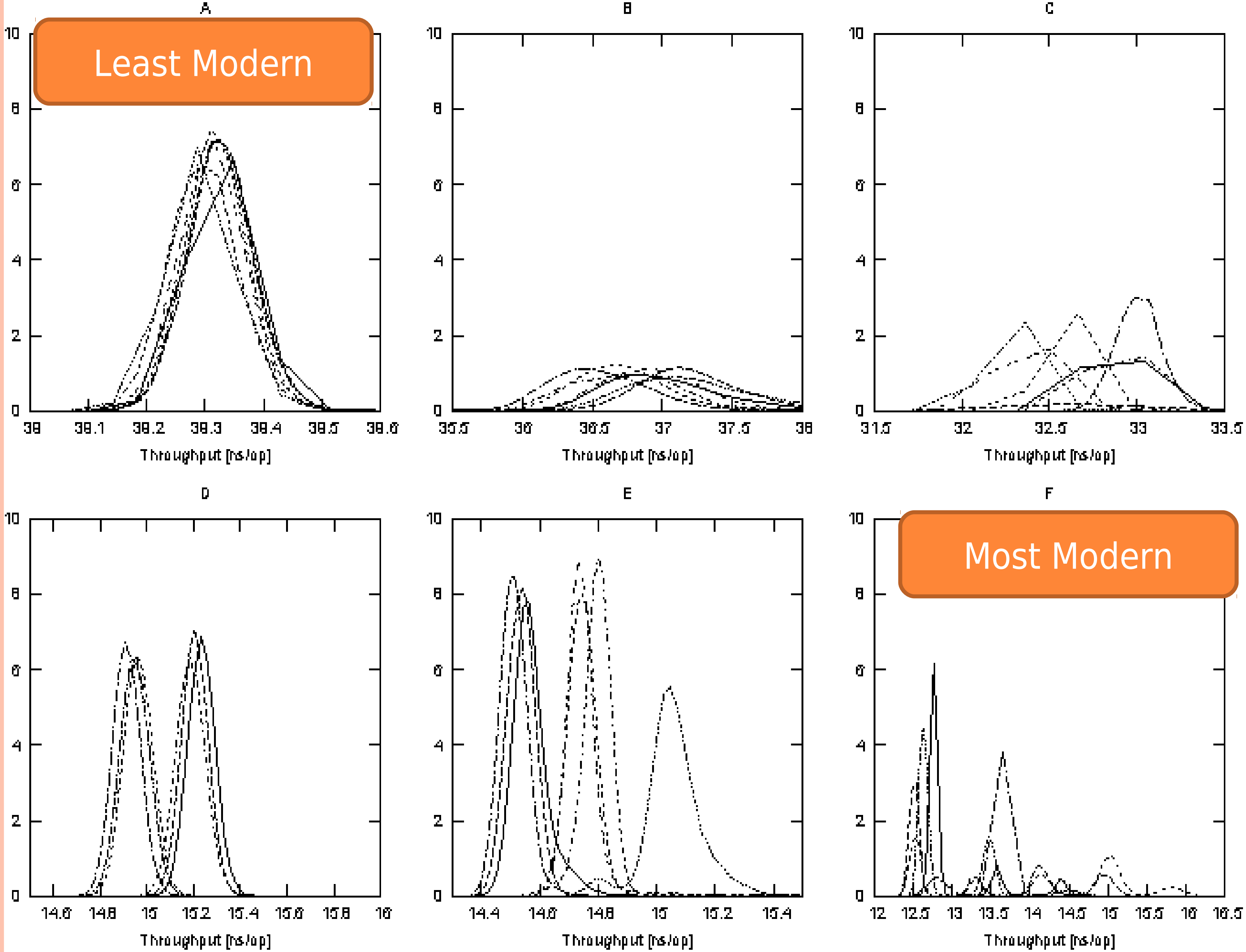


Probably some (small) correlation between results

SITUATION IS WORSE IF JIT IS DISABLED



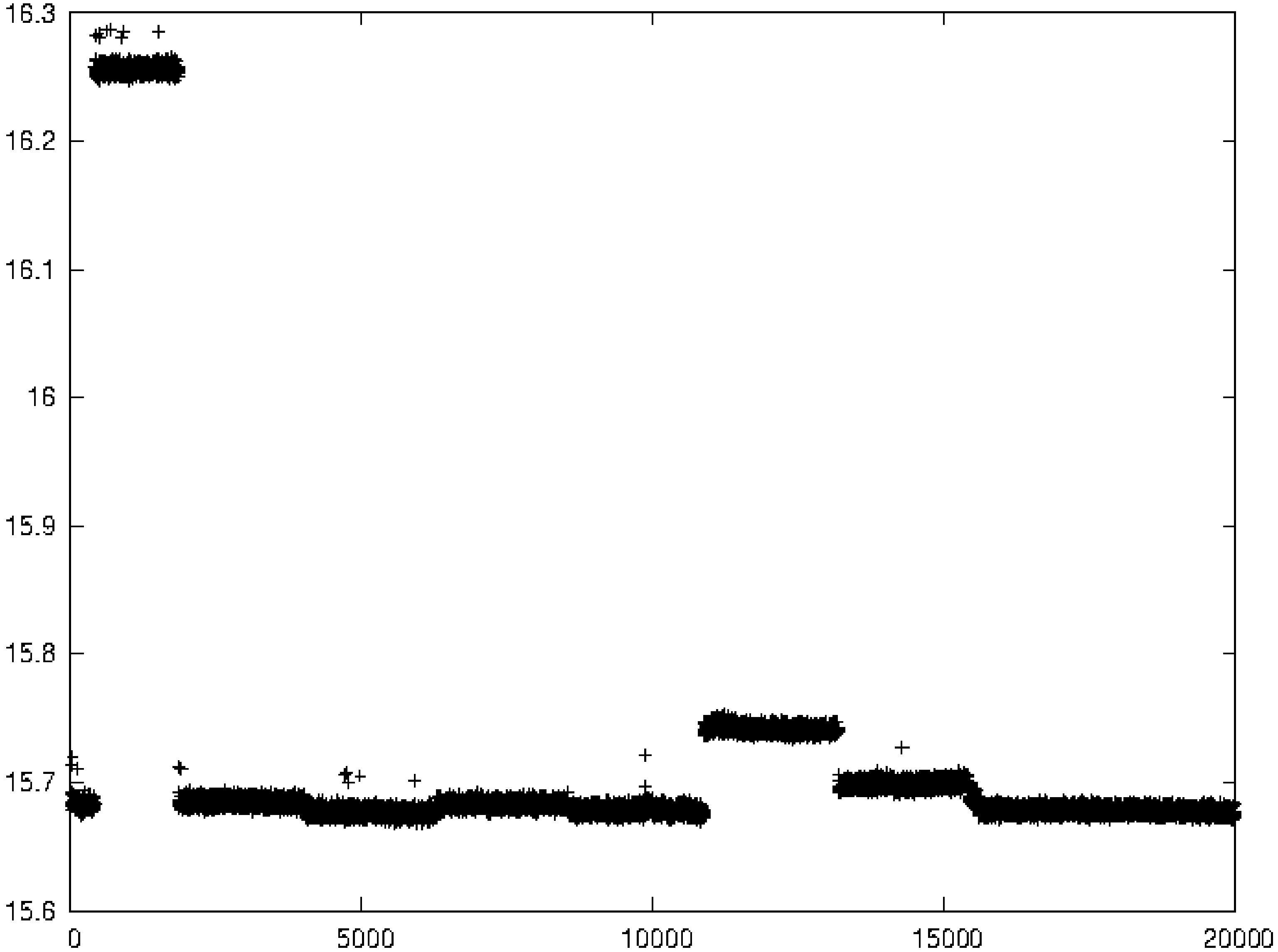
EXPLORE OTHER COMPUTING PLATFORMS



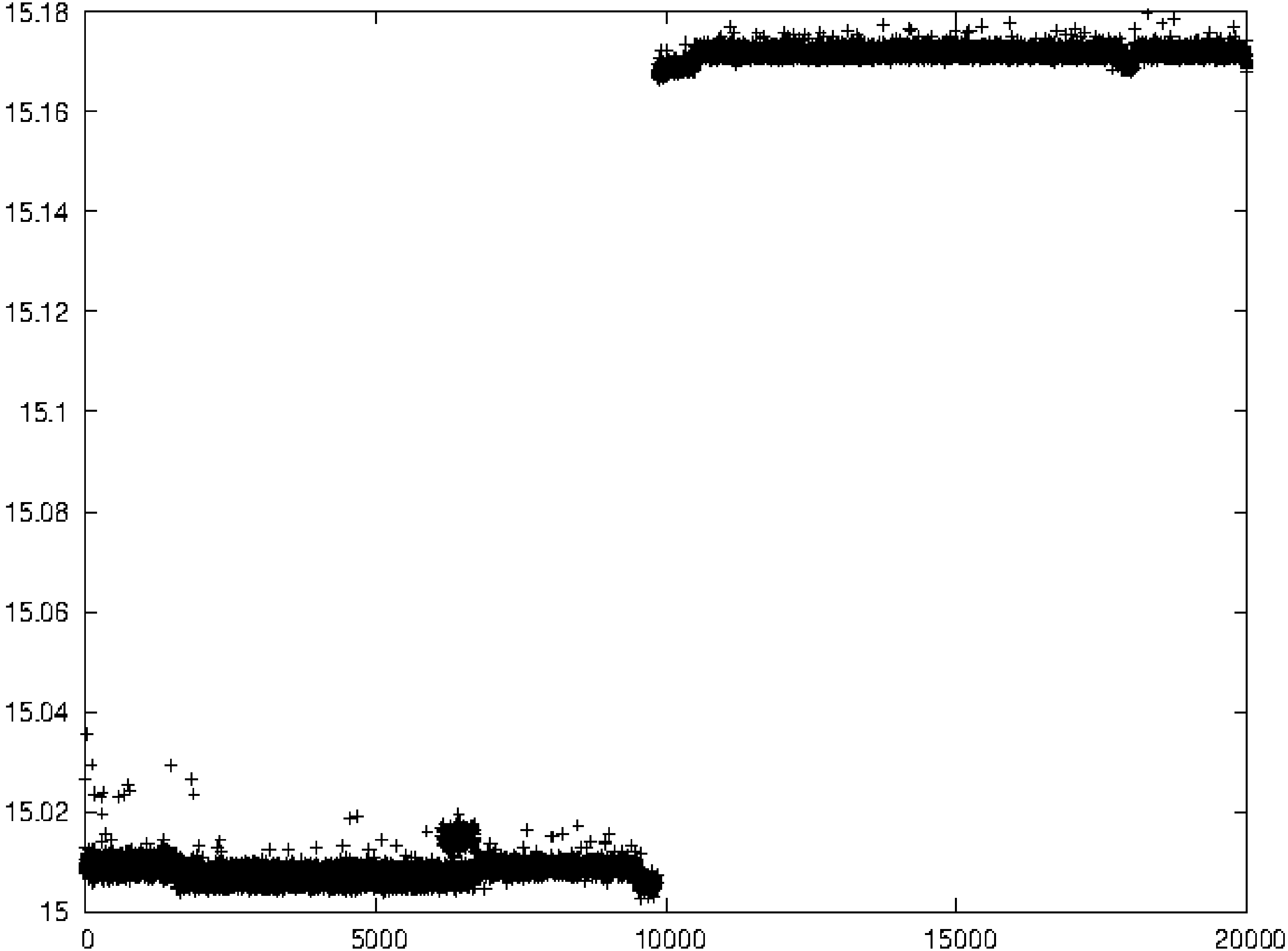
VARIATION ON BENCHMARKED CODE

- Variations:
  - Array Bubble Sort
  - List Bubble Sort
  - XOR of Random Values
  - Ergodic List
- Results:
  - Essentially the same
  - XOR-Random is slighlty different:
    - Distinct peaks
    - Measurements in a single invocation are far from a normal distribution

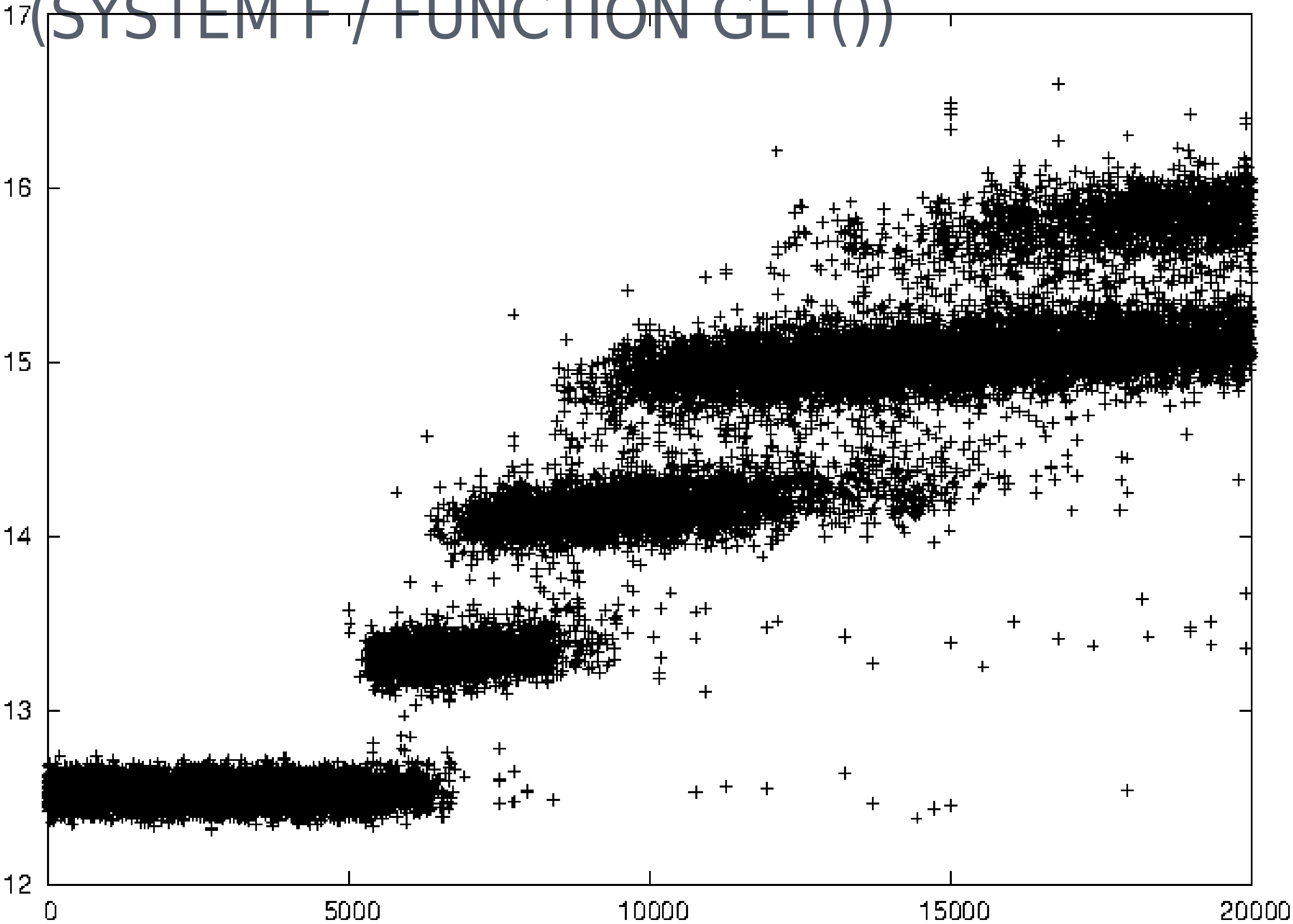
MULTIPLE STEADY STATE IN LIST BUBBLE SORT



MULTIPLE STEADY STATES IN XORRANDOMS

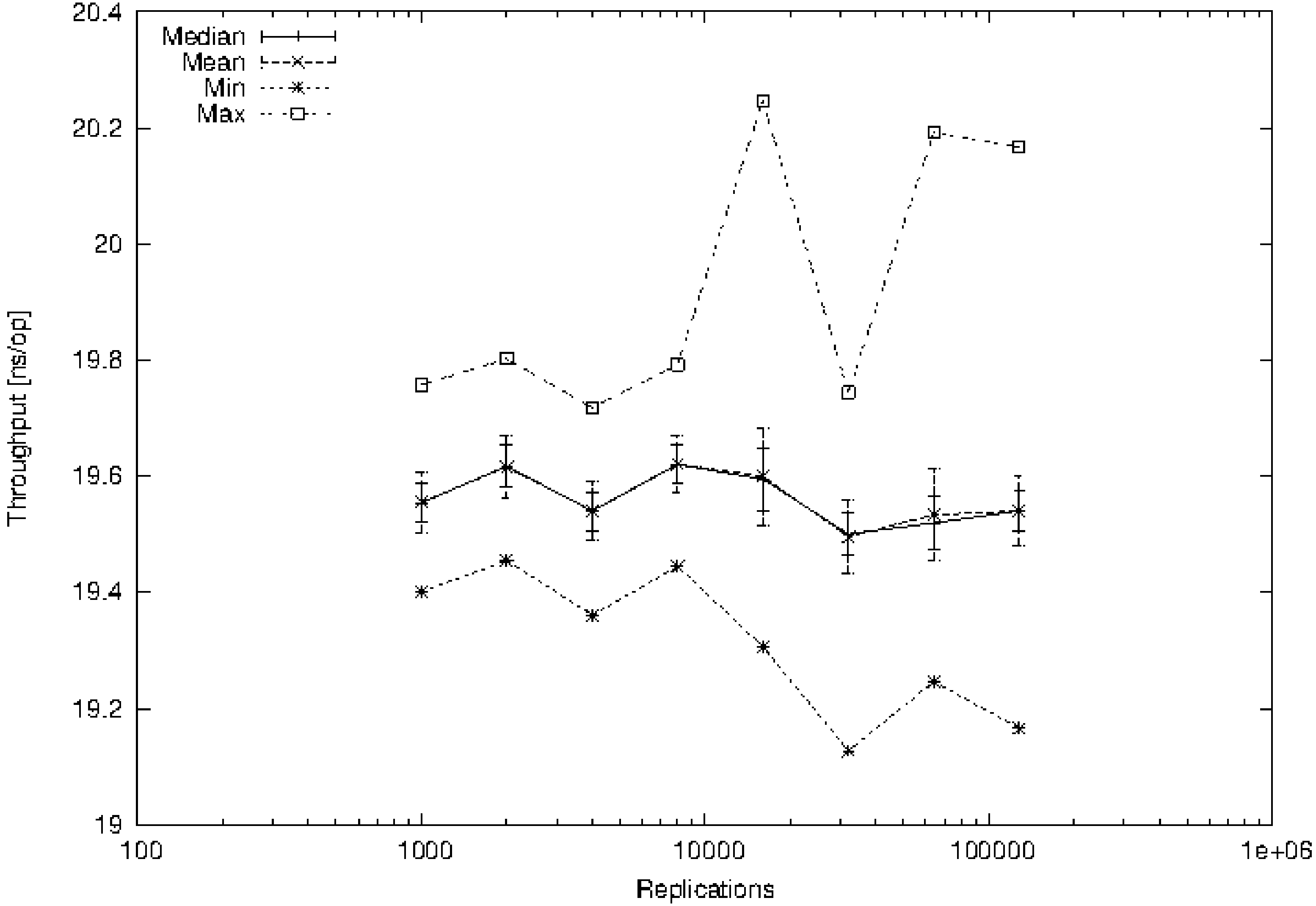


CONCURRENT MULTIPLE STEADY STATES (SYSTEM F / FUNCTION GET())



NO RECOVERY FROM PRLOGUE RUNS

FUNCTION GET NEVER REACHES THE 15NS PERFORMANCE, AFTER JUST 50 FOREIGN CALLS



CONCLUSIONS?

- Questions!
  - Did we miss some ghost threads?
  - Why?
  - Statistical methods for compensations?
  - Is the trouble of micro-benchmarking worth the dubious results?
- On the more positive side:
  - Statics is still valid: if you do many invocations, you obtain some estimate on the mean and on the SD.
  - Compilation planning is probably at fault of the JIT memory.
- Further research:
  - Spend many more months exploring the exponential parameter space.