PRNG benchmarks (version 2)

S. Gerlach

Jan 31, 2018

PRNG Tests

- Uniform [0,1] and standard normal distributed PRNGs
- Run time for calculating average of 10^9 random numbers (double precision)
- Tested on standard workstation: i5-6500 CPU (Skylake)
  - GSL (system, 2.4-gnu, 2.4-intel)
  - C++-11, Boost, TRNG
  - MKL
- Parallel PRNG (TRNG, MKL)

GSL benchmarks

PRNGs tested:

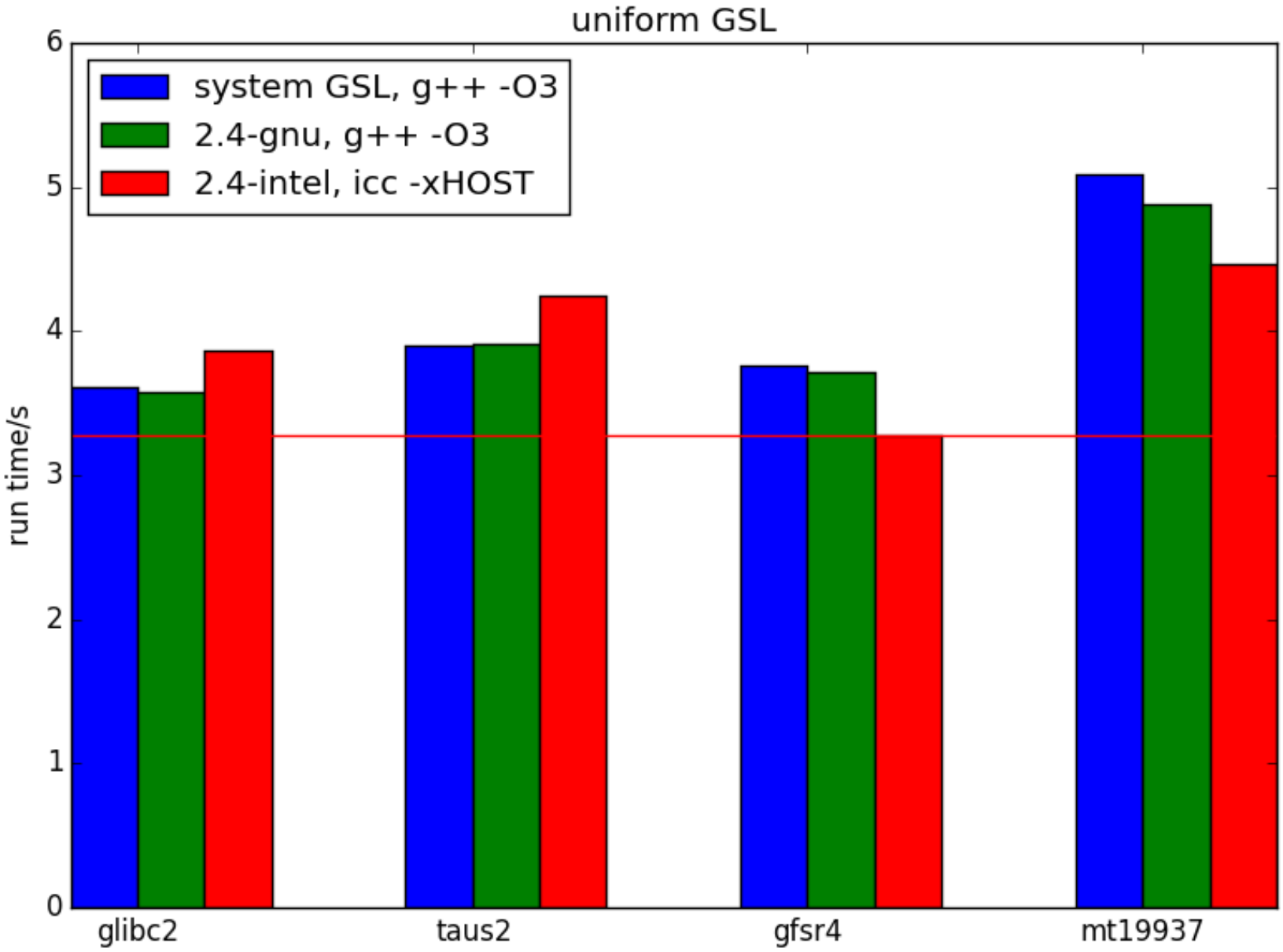
- **random()** (`gsl_rng_random_glibc2`)
- **Tausworthe** (`gsl_rng_taus2`, 870 k doubles/sec*)
- **GFSR** (`gsl_rng_gfsr4`, 855 k doubles/sec)
- **MT19937** (`gsl_rng_mt19937`, 769 k doubles/sec)

Normal distribution:

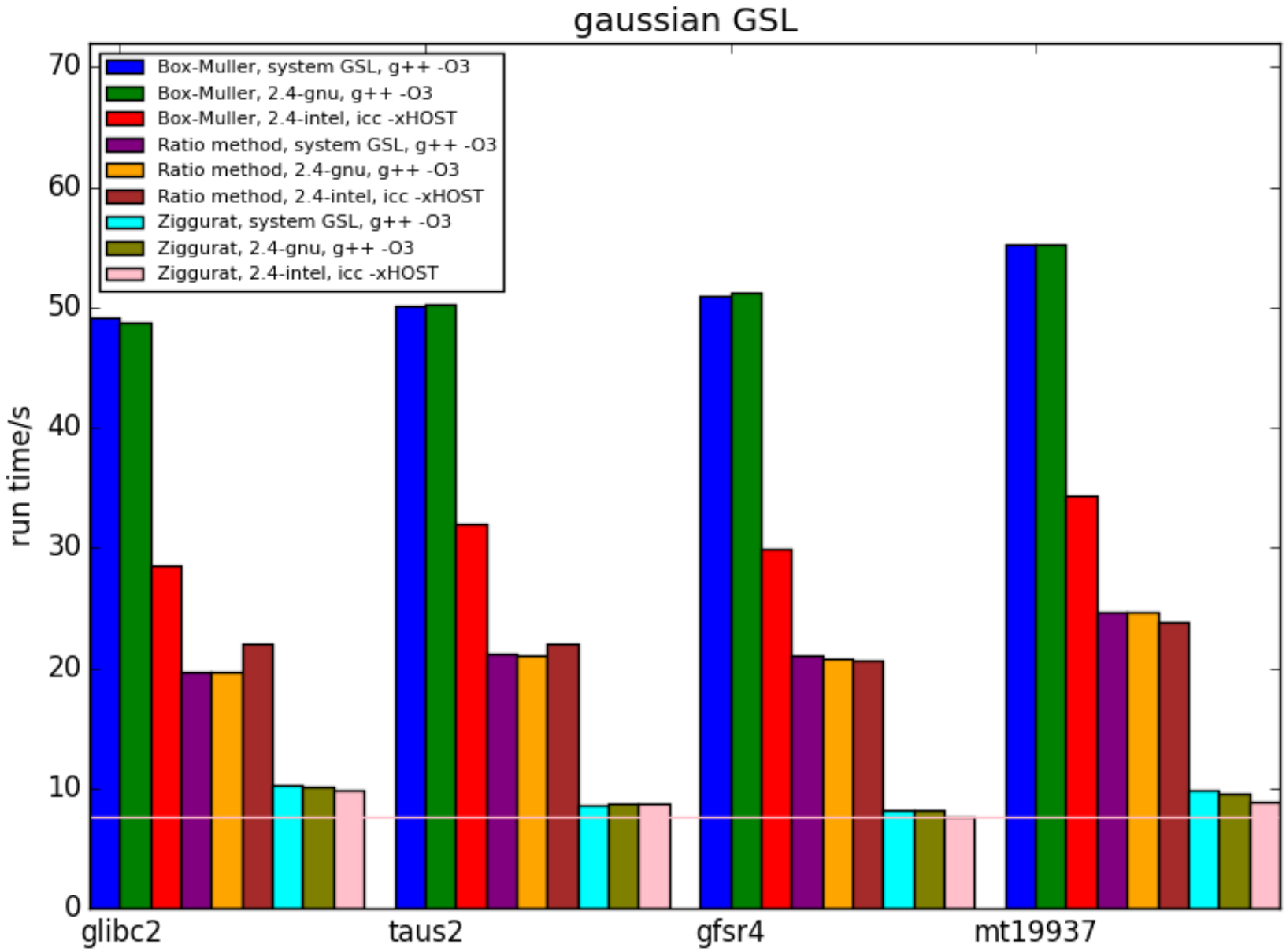
- Box-Muller (`gsl_ran_gaussian()`)
- Ratio method (`gsl_ran_gaussian_ratio_method()`)
- Zigurrat method (`gsl_ran_gaussian_ziggurat()`)
- 3 point method (own implementation)

* From GSL website

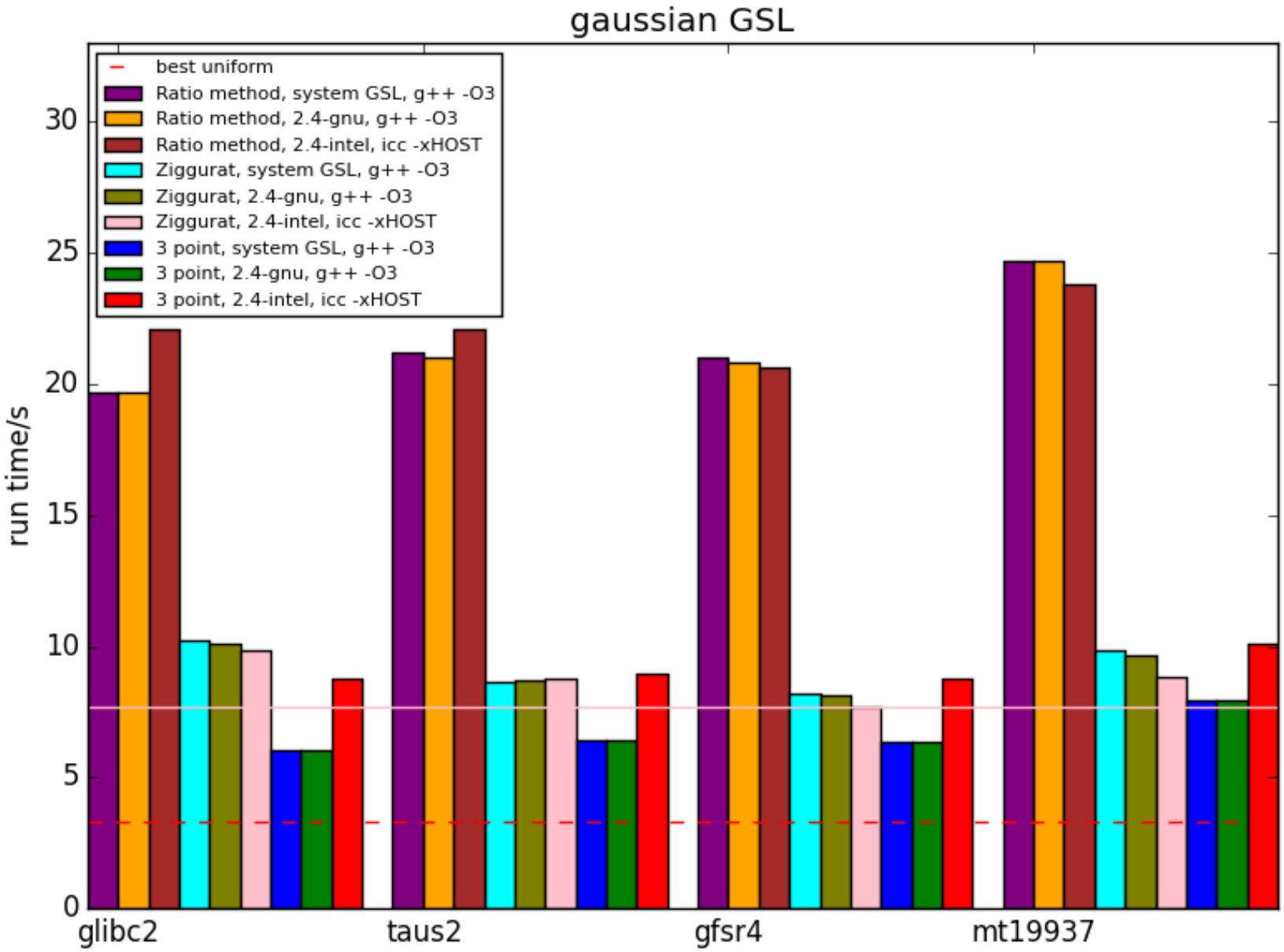
GSL benchmarks



GSL benchmarks



GSL benchmarks



GSL benchmark results

- **uniform**: GSL (2.4-intel) with gfsr4: 3.28 s
- **gaussian**: GSL (2.4-intel) ziggurat with gfsr4: 7.68 s
- fastest 3 point gaussian:
  6.02 s (system GSL, random()),
  6.33 s (system GSL, gfsr4)

C++11 & libraries benchmarks

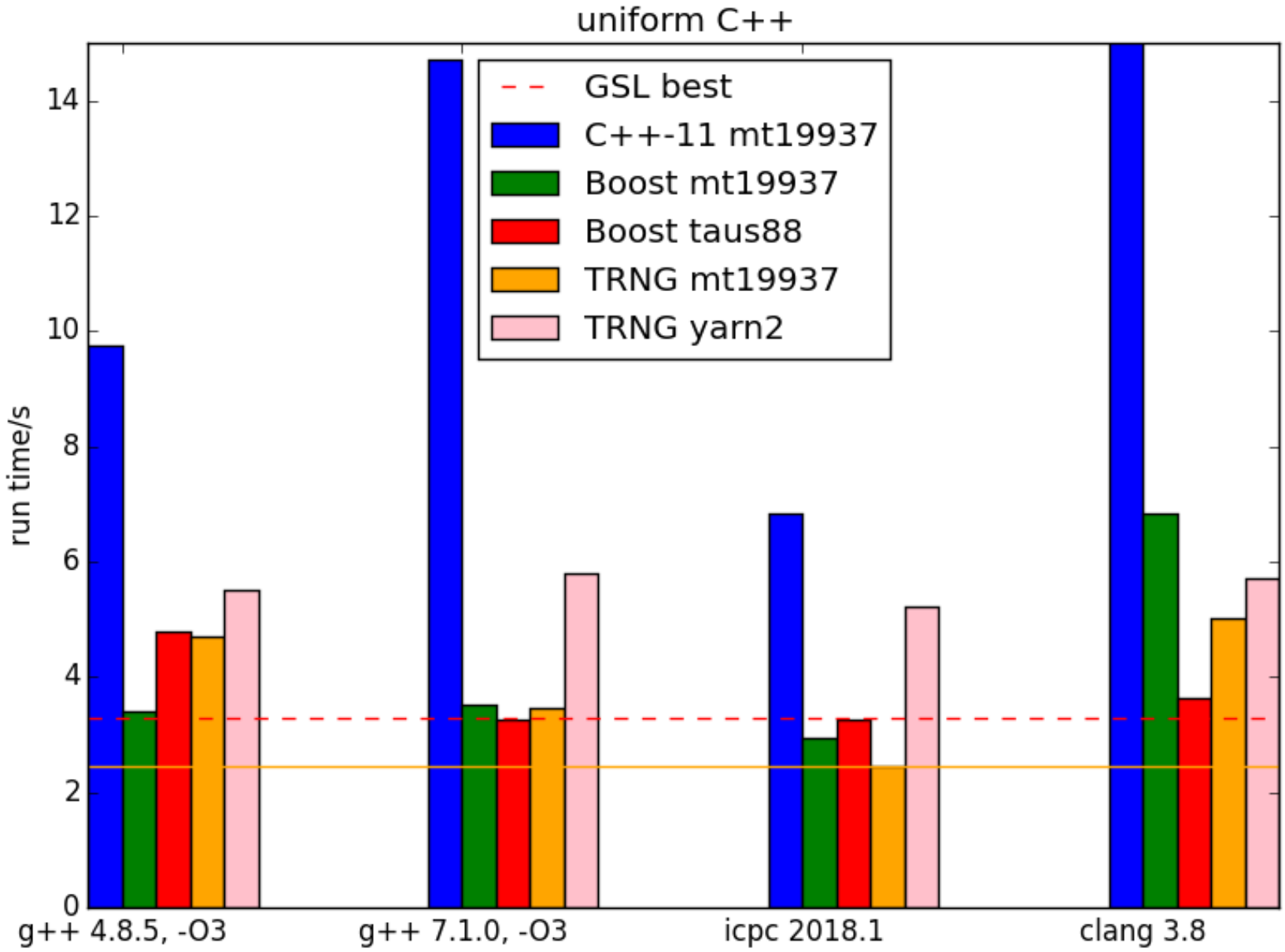
PRNGs tested:

- **C++11** (mt19937)
- **Boost** 1.54 (taus88, mt19937)
- **TRNG** 4.19-intel (yarn2, mt19937)

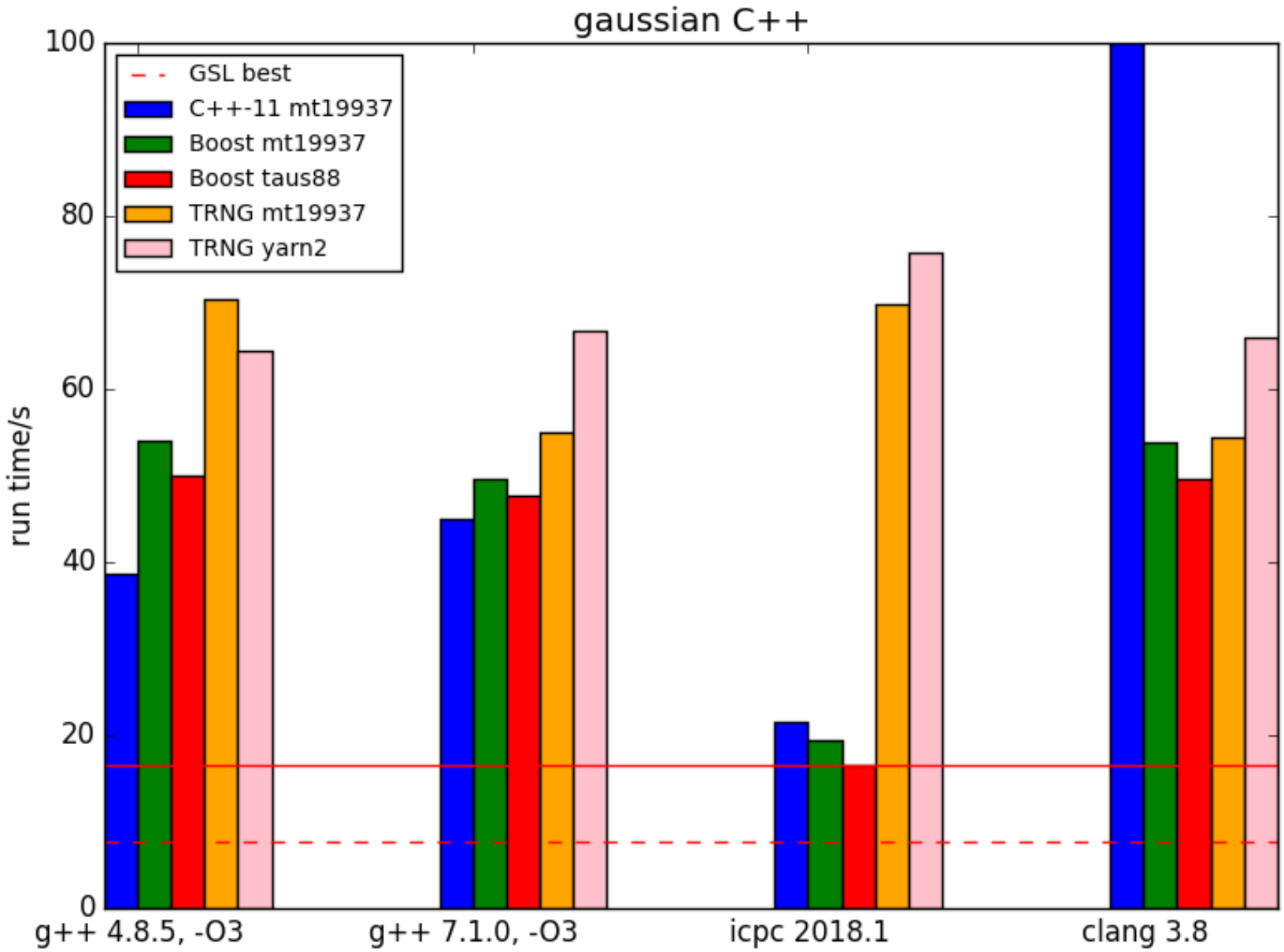
Compiler used:

- `g++ 4.8.5, -O3`
- `g++ 7.1.0, -O3`
- `icpc 2018.1`
- `clang 3.8`

C++11 & libraries benchmarks



C++11 & libraries benchmarks



C++11 & libraries benchmark results

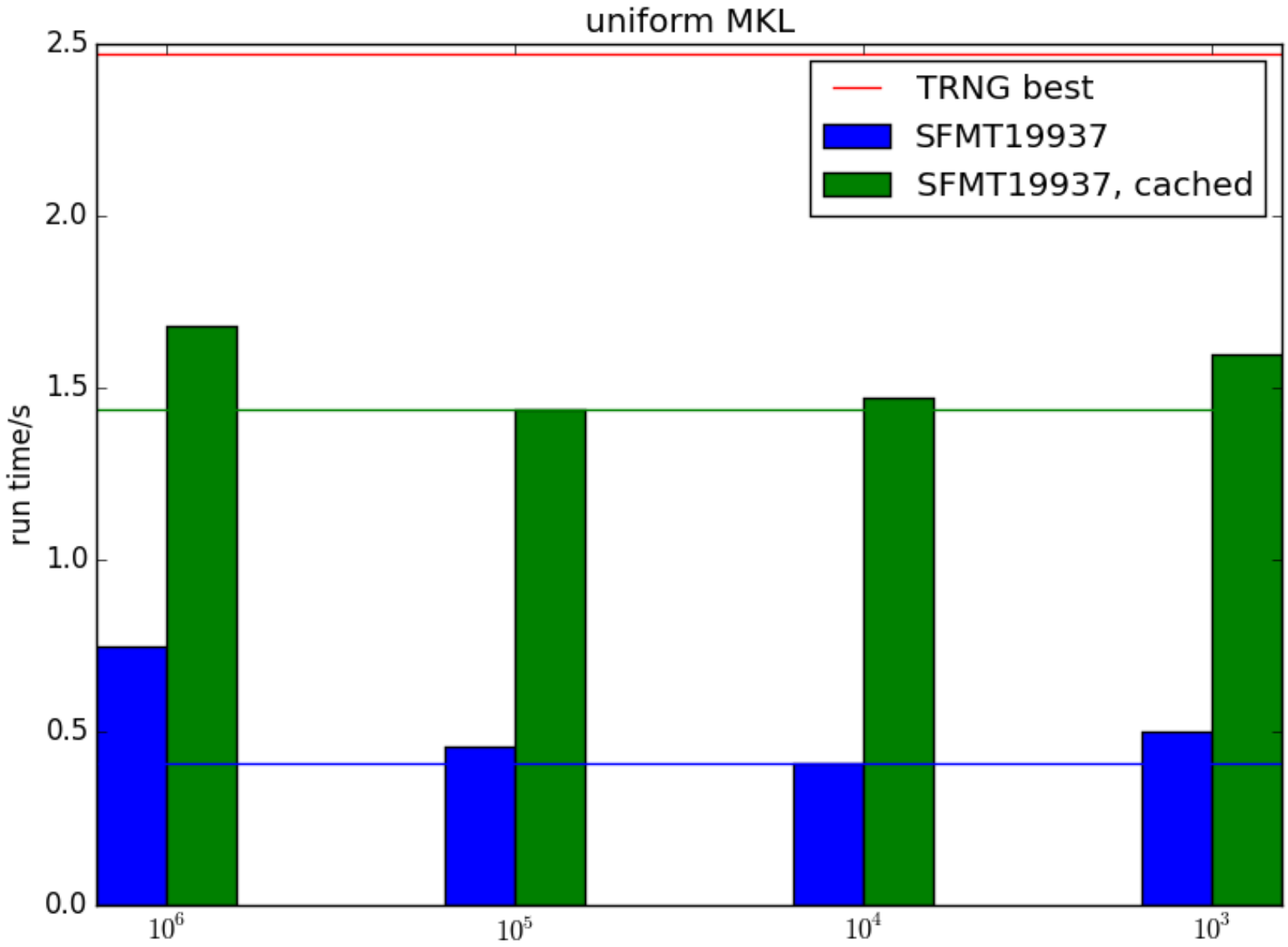
- **uniform**: TRNG with icpc (mt19937): 2.46 s
- **gaussian**: GSL is faster

MKL benchmarks

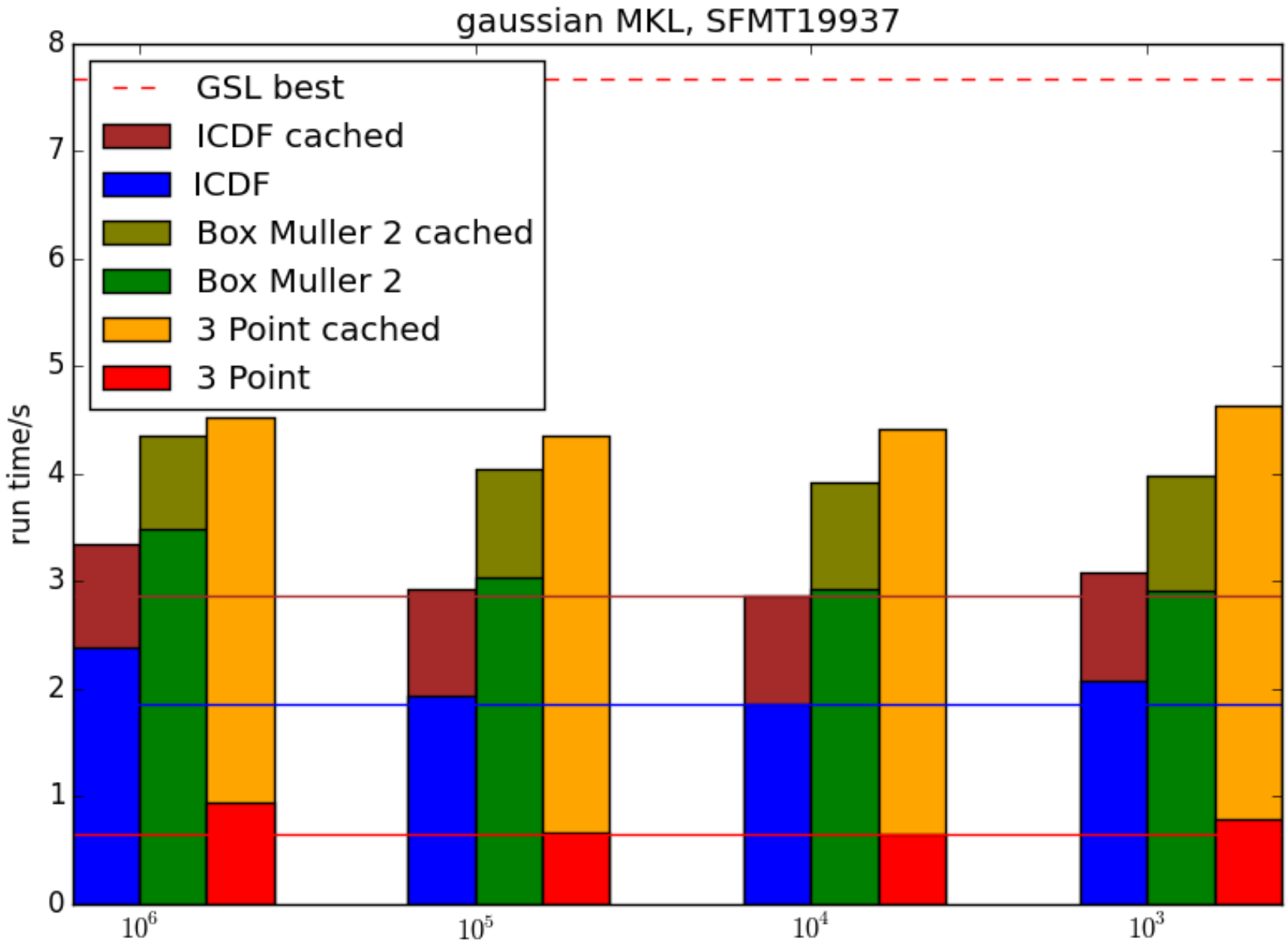
PRNG:

- **MKL** 2018u1
- generates vector of random numbers (size=10^6, 10^5, 10^4, 10^3)
- SFMT19937 (SIMD-oriented Fast MT): fastest generator
- conversion method: ICDF, BOXMULLER2, 3Point

MKL benchmarks



MKL benchmarks



MKL benchmark results

- **uniform**: MKL SFMT19937, ICDF (10^4 size)
  0.41 s (cached: 1.44 s)
- **gaussian**: MKL SFMT19937, ICDF (10^4 size)
  1.85 s (cached: 2.87 s)

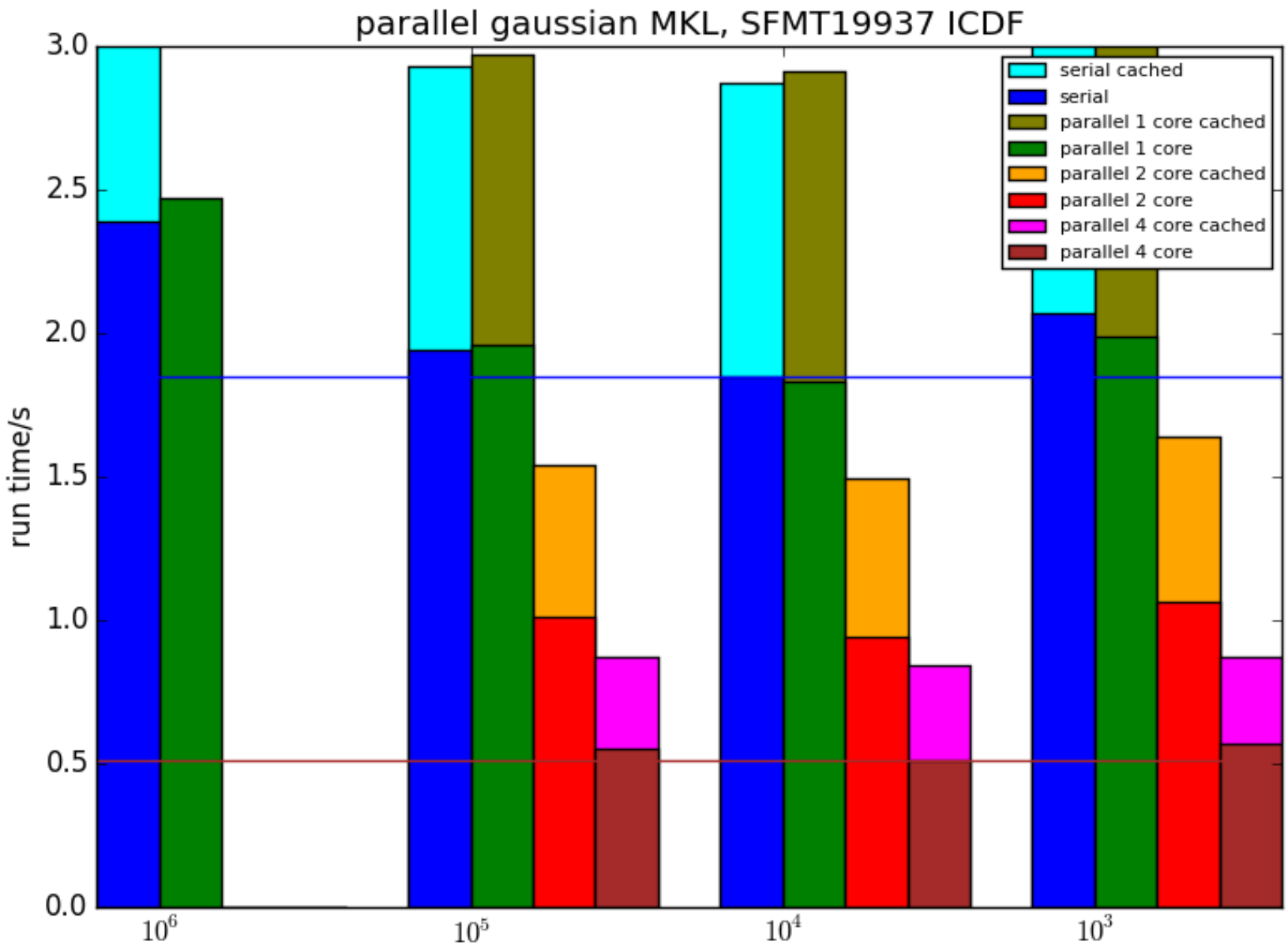
Parallel RNG

Strategies:

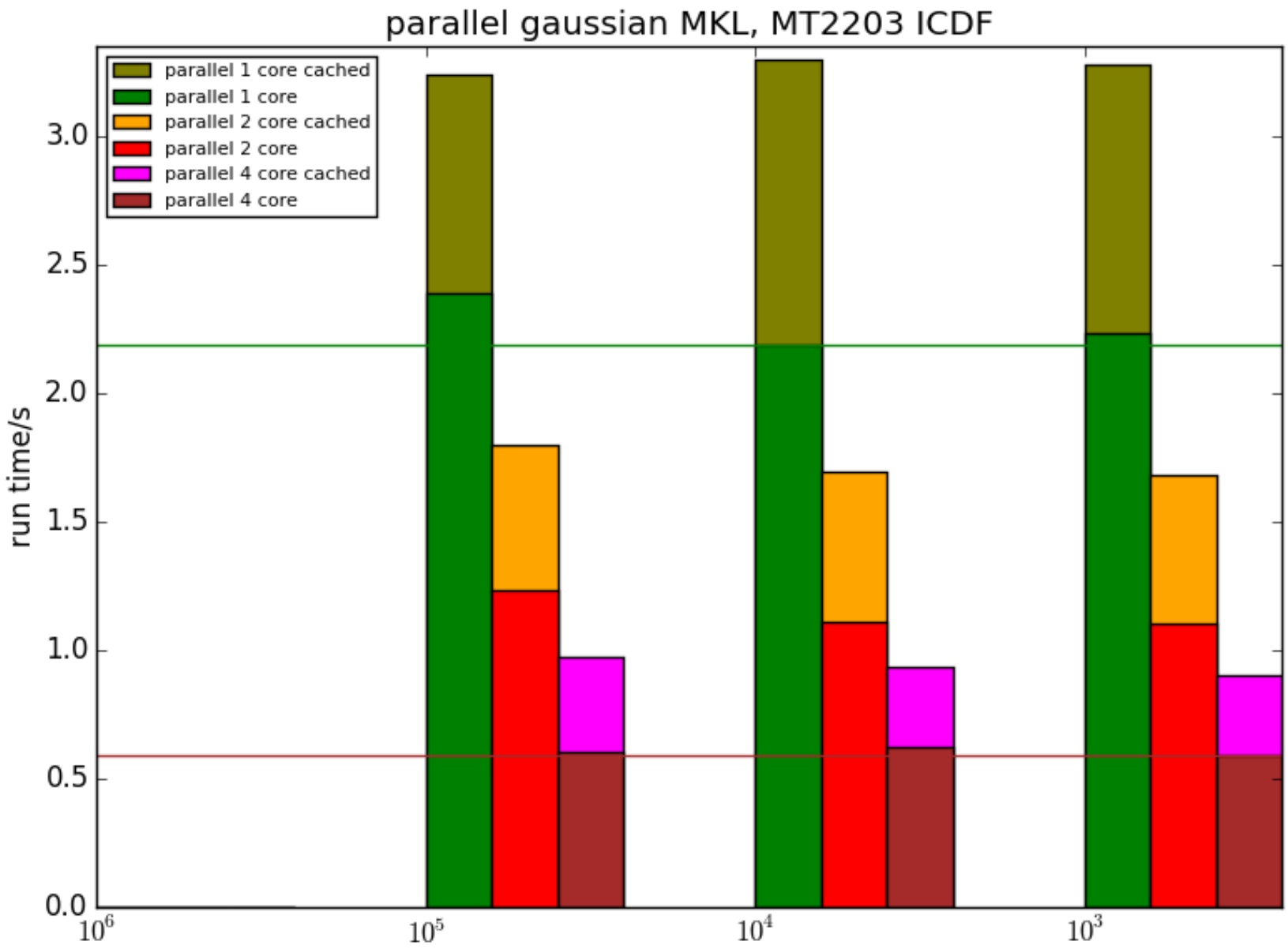
- random seed/ sequential seed: unknown behaviour (overlapping sequences)
- parameterization: depends on RNG, unpractical
- block-splitting: only when N is known
- leap-frogging: when supported by RNG
- independent streams: when supported by RNG

- MKL - SMT19937: only block-splitting
- MKL - MT2203: independent streams!
- TRNG - yarn2: leap-frogging

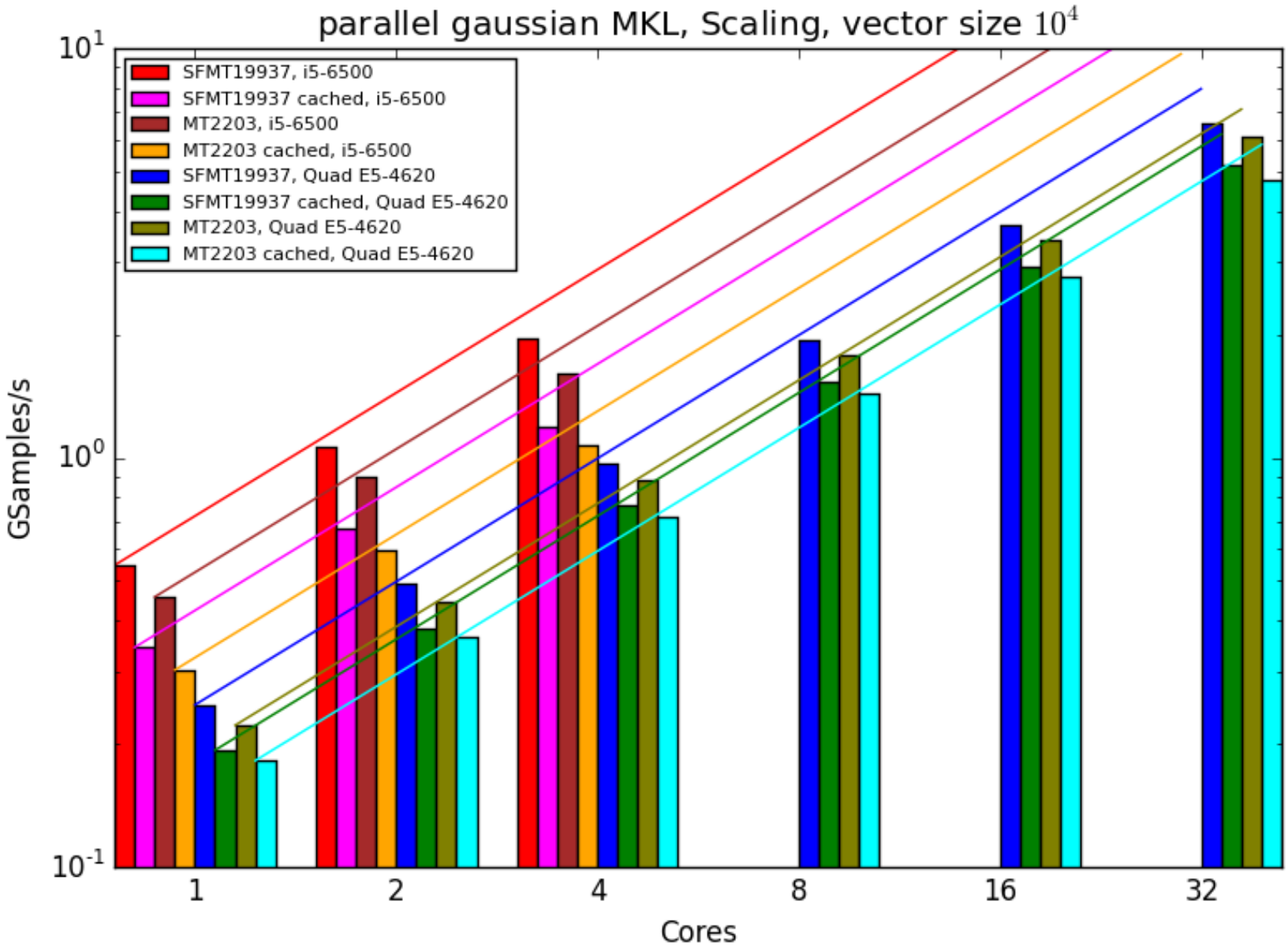
Parallel results - MKL - SFMT19937 Block splitting



Parallel results - MKL - MT2203 independent streams



Parallel results - MKL - Skaling



Summary

Generating 10^9 random numbers (double precision):

- Fastest **GSL** (2.4-intel, gfsr4, Ziggurat):
  3.28 s (uniform)
  7.68 s (gaussian)
- **TRNG** 1.19 (Intel 2018.1, mt19937):
  2.47 s (uniform)
- **MKL** 2018.1 (SFMT19937, 10^4):
  0.41 s|cached: 1.44 s (uniform)
  1.85 s|cached: 2.87 s (gaussian)
- MKL 2018.1 **parallel** (SFMT19937, 10^4, 4 Cores):
  0.51 s|cached: 0.84 s (gaussian), very good scaling
- TRNG **parallel** scaling good, but gaussian already very slow on 1 core

Code - 3 point rule

```
#define SQRT3 1.732050807568877293527446341511

inline double threep(double v) {
        if (v < 1./6.)
                return -SQRT3;
        else if (v > 5./6.)
                return SQRT3;

        return 0;
}
```

Code - random number caching

```
#define NR 10000
VSLStreamStatePtr stream;
double rcache[NR];

inline double ran() {
        static int index = NR;

        if (index == NR) {
                vdRngGaussian(VSL_RNG_METHOD_GAUSSIAN_ICDF,
                        stream, NR, rcache, 0.0, 1.0);
                index = 0;
        }
        return rcache[index++];
}
```

Code - parallel

```
#include <stdio.h>
#include <mkl_vsl.h>
#include <omp.h>
#define N 1e10  // random numbers to generate
#define M 10000 // vector size

int main() {
      int n = omp_get_max_threads(); omp_set_num_threads(n);

      VSLStreamStatePtr stream[n];
      for (int i = 0; i < n; i++)
            vslNewStream(&stream[i], VSL_BRNG_MT2203 + i, 4711);

      double sum = 0.;
#pragma omp parallel for reduction(+:sum)
      for (int i = 0; i < n; i++) {
            double rcache[M];
            int index = M;
            for (long j = 0; j < N/n; j++) {
                  if (index == M) {
                        vdRngGaussian(VSL_RNG_METHOD_GAUSSIAN_ICDF, stream[i], M, rcache, 0.0, 1.0);
                        index = 0;
                  }
                  sum += rcache[index++];
           }
      }
      printf("sum = %g\n", sum/(double)N);

      for (int i = 0; i < n; i++)
            vslDeleteStream(&stream[i]);
      return 0;
}
```

Links

https://www.gnu.org/software/gsl/manual/html_node/Random-Number-Generation.html
https://www.gnu.org/software/gsl/manual/html_node/Random-Number-Generator-Performance.html
http://en.cppreference.com/w/cpp/numeric/random
https://www.numbercrunch.de/trng/
https://software.intel.com/en-us/mkl-developer-reference-c-random-number-generators
https://software.intel.com/en-us/node/590367