# Fast Special Functions

Sage Days 35: Algorithms in Number Theory and FLINT
University of Warwick

Fredrik Johansson

RISC, Linz

2011-12-17

## Why FLINT and special functions?

Applications: algorithms for numerical and symbolic summation, integration, ODEs, combinatorics...

Asymptotically fast **polynomial** and **power series arithmetic** over $\mathbb{Z}$, $\mathbb{Z}/p\mathbb{Z}$ and $\mathbb{Q}$.

Highly optimized modular arithmetic, integer vectors...

Nice **development framework** (modular and well-tested codebase, automagical build/test/documentation system).

## Special functions currently provided in FLINT

**Power series** expansions of elementary function (sqrt, exp, log, sin, asin, tan, atan ...)

**Number sequences**: harmonic numbers, primorials, Bernoulli $B_n$, Euler $E_n$, Stirling $s(n,k)$, $S(n,k)$, Bell $B_n$, partitions $p(n)$, $\Delta$-function $q$-expansion, divisor sum, Euler $\varphi$, Möbius $\mu$, Dedekind sums ...

Some special **polynomials**: cyclotomic, Legendre, Chebyshev, Bernoulli, Swinnerton-Dyer ...

High-precision numerical evaluation: $\pi$, $\gamma$, $\zeta(n)$ (otherwise: MPFR)

## Example: Bell numbers (set partitions)

$$\sum_{n=0}^{\infty} \frac{B_n}{n!} x^n = \exp\left(e^x - 1\right)$$

$$\{B_n\}_{n=0}^{\infty} = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$$

## Classical algorithm: triangular recurrence

$$\begin{array}{lllll}
1 \\
1 & 2 \\
2 & 3 & 5 \\
5 & 7 & 10 & 15 \\
15 & 20 & 27 & 37 & 52 \\
\ldots
\end{array}$$

$O(n^2)$ additions

$\tilde{O}(n^3)$ time complexity, $\tilde{O}(n^2)$ space complexity

Gives $B_0, \ldots B_n$ very efficiently in practice at least for small $n$. As far as I know, this algorithm (or something equivalent) is the only thing used in Sage, GAP, Mathematica, ...

## Computing just the *n*th Bell number

If we only want a single value, it is much better to use a series representation:

$$B_n = e^{-1} \sum_{k=0}^{N} \frac{k^n}{n!} + \varepsilon(N)$$

$N \approx n$

$\tilde{O}(n^2)$ time complexity, $\tilde{O}(n)$ space complexity

We can save time using *binary splitting* to amortize the factorials. But we still effectively need to compute all the powers $2^n, 3^n, \ldots N^n$.

## An alternative way to compute the *n*th Bell number

$$B_n = \sum_{k=0}^{n} \left\{ {n \atop k} \right\} = \sum_{k=0}^{n} \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n = \sum_{k=0}^{n} \frac{(n-k)^n}{(n-k)!} \sum_{k=0}^{n-k} \frac{(-1)^j}{j!}$$

This is not any better.

However, we might save a constant factor using a **multimodular** algorithm: evalute the sum modulo several word-size primes and reconstruct $B_n$ using the Chinese Remainder Theorem.

Possible speedup depends on the relative speed of bignum and single-word arithmetic. We gain from a smaller total bit size ($\log_2 B_n$ bits instead of $\log_2 n! B_n$), and ability to sieve out small multiples from the powers.

## Asymptotically fast vector computation of Bell numbers

$$\sum_{n=0}^{\infty} \frac{B_n}{n!} x^n = \exp\left(e^x - 1\right)$$

FLINT provides an asymptotically fast power series exponential $(O(M(n)))$.

This algorithm is **quasi-optimal**: $\tilde{O}(n)$ modulo a fixed prime $p$, $\tilde{O}(n^2)$ using arithmetic over $\mathbb{Q}$.

But overhead is higher than other methods for small $n$.

Using a multimodular algorithm (with fast CRT reconstruction) is particularly attractive since we can clear denominators while still in the modular domain.

## Computing Bell numbers mod $p$

| $n$ | Triangular | Power series | Single value |
|---|---|---|---|
| $10^3$ | 2 ms | 4 ms | 0.1 ms |
| $10^4$ | 220 ms | 71 ms | 1 ms |
| $10^5$ | 22 s | 1 s | 10 ms |
| $10^6$ | 2400 s | 15 s | 270 ms |

$$p = 2^{63} + 29$$

## Computing the first $n$ Bell numbers (over $\mathbb{Z}$)

| $n$ | Triangular | Series (mmod) | Series (rational) | bsplit |
|---|---|---|---|---|
| 1000 | 0.09 s | 0.28 s | 0.91 s | 2.9 s |
| 2000 | 1.35 s | 2.57 s | 5.55 s | 33 s |
| 4000 | 11.6 s | 14.6 s | 33 s | 398 s |
| 6000 | 40.8 s | 38.6 s | 93 s | |
| 8000 | 99 s | 75 s | 196 s | |
| 10000 | 197 s | 129 s | | |
| 20000 | 1701 s | 642 s | | |

## Computing just the $n$th Bell number (over $\mathbb{Z}$)

| $n$ | Binary splitting | Multimodular |
|---|---|---|
| $10^3$ | 9 ms | 19 ms |
| $10^4$ | 3.6 s | 2.4 s |
| $10^5$ | 970 s | 370 s |
| $2 \times 10^5$ | 4670 s | 1920 s |

## Wilf's conjecture

The *complementary Bell numbers* or *Rao Uppuluri-Carpenter numbers* are defined by

$$\sum_{n=0}^{\infty} \frac{c_n}{n!} x^n = \exp\left(1 - e^x\right)$$

$$\{c_n\}_{n=0}^{\infty} = 1, -1, 0, 1, 1, -2, -9, -9, 50, 267, \ldots$$

Wilf conjectured that $c_2$ is the only zero.

It has been proved that at most one more zero exists.

## Sample computation with FLINT

```
nmod_poly_t b;
nmod_poly_init(b, mod);
nmod_poly_set_coeff_ui(b, 1, 1);
nmod_poly_exp_series(b, b, n+1);
nmod_poly_neg(b, b);
nmod_poly_set_coeff_ui(b, 0, 0);
nmod_poly_exp_series(b, b, n+1);
```

Verification of Wilf's conjecture up to $n = 10^9$ working mod $p = 10^{12} + 39$: 5 hours on 1 CPU

## Computation of special numbers in FLINT

Most of this applies to Bernoulli numbers, Euler numbers, etc. In general, we want to use several algorithms:

**Basecase**: table lookup, recurrences.

**Vector computation**: power series arithmetic (possibly multimodular).

**Single coefficients**: analytic formulas (e.g. Dirichlet series for Bernoulli and Euler numbers), multimodular algorithms.

Improved algorithms for Bernoulli numbers by David Harvey (*bernmm* is faster than the zeta algorithm currently in FLINT above $n \approx 10^6$).

## Integer partitions

And now for something completely different . . .

## The partition function

The number of *integer* partitions $p(n)$ is given by Euler's generating function

$$\sum_{n=0}^{\infty} p(n) x^n = \prod_{k=1}^{\infty} \frac{1}{1-x^k}$$

$$\{p(n)\}_{n=0}^{\infty} = 1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42, \ldots$$

Size of $p(n)$: $\approx n^{1/2}$ digits

Fast evaluation allows investigating the distribution of $p(n) \bmod m$.

## Asymptotically fast vector computation of $p(n)$

$$\sum_{n=0}^{\infty} p(n)x^n = \prod_{k=1}^{\infty} \frac{1}{1-x^k} = \left( \sum_{k=-\infty}^{\infty} (-1)^k x^{k(3k-1)/2} \right)^{-1}$$

We only need a single power series inversion (very fast in FLINT!)

$\tilde{O}(n^{3/2})$ time complexity over $\mathbb{Z}$

$\tilde{O}(n)$ time complexity over $\mathbb{Z}/m\mathbb{Z}$ for fixed $m$

## Isolated values – the state of the art

Jonathan Bober wrote a new partition function for Sage in 2007.

```
Mathematica: Timing[PartitionsP[10^7];]
{2.42, Null}

sage: %time number_of_partitions(10^7, algorithm='pari');
Wall time: 4.38 s

sage: %time number_of_partitions(10^7, algorithm='bober');
Wall time: 0.39 s
```

All three systems use the *Hardy-Ramanujan-Rademacher* formula. How fast can we make it?

## The Hardy-Ramanujan-Rademacher formula

Hardy and Ramanujan (1917), Rademacher (1936)

$$p(n) = \frac{1}{\pi\sqrt{2}} \sum_{k=1}^{\infty} \sqrt{k}\, A_k(n)\, \frac{d}{dn} \left( \frac{1}{\sqrt{n-\frac{1}{24}}} \sinh \left[ \frac{\pi}{k} \sqrt{\frac{2}{3}\left(n - \frac{1}{24}\right)} \right] \right)$$

$$A_k(n) = \sum_{0 \le h < k;\ \gcd(h,k)=1} e^{\pi i \left[ s(h,k) - \frac{1}{k} 2nh \right]}$$

$$s(h,k) = \sum_{i=1}^{k-1} \frac{i}{k} \left( \frac{hi}{k} - \left\lfloor \frac{hi}{k} \right\rfloor - \frac{1}{2} \right)$$

## Complexity of numerical evaluation

Odlyzko (1995): the HRR formula "gives an algorithm for calculating $p(n)$ that is close to optimal, since the number of bit operations is not much larger than the number of bits of $p(n)$". But no further details (let alone concrete algorithms) are to be found in the literature.

We need $n^{1/2}$ terms calculated to a precision of $b_k = O(n^{1/2}/k + \log n)$ bits to determine $p(n)$.

Assume that we can evaluate each term in quasilinear time $O(b \log^c b)$. Then the total cost is quasioptimal

$$\sum_{k=1}^{n^{1/2}} \left(\frac{n^{1/2}}{k}\right) \log^c \left(\frac{n^{1/2}}{k}\right) \sim n^{1/2} \log^{c+1} n$$

## Dedekind sums

The "inner loop" consists of the Dedekind sum

$$s(h,k) = \sum_{i=1}^{k-1} \frac{i}{k} \left( \frac{hi}{k} - \left\lfloor \frac{hi}{k} \right\rfloor - \frac{1}{2} \right)$$

Naive algorithm: $O(k)$ integer or rational operations

$O(k^2)$ integer operations to evaluate $A_k(n)$

$O(n^{3/2})$ integer operations to evaluate $p(n)$

We can compute $p(n)$ in quasi-optimal time if we get $A_k(n)$ down to $O(\log^c k)$ function evaluations and integer operations

## Fast computation of Dedekind sums

Let $0 < h < k$ and let $k = r_0, r_1, \ldots, r_{m+1} = 1$ be the sequence of remainders in the Euclidean algorithm for $\gcd(h, k)$. Then

$$s(h,k) = \frac{(-1)^{m+1} - 1}{8} + \frac{1}{12} \sum_{j=1}^{m+1} (-1)^{j+1} \frac{r_j^2 + r_{j-1}^2 + 1}{r_j r_{j-1}}.$$

Can be implemented directly or with the fraction-free algorithm of Knuth (1975).

$O(\log k)$ integer or rational operations to evaluate $s(h,k)$

$O(k \log k)$ integer operations to evaluate $A_k(n)$

$O(n \log n)$ integer operations to evaluate $p(n)$

## Evaluating $A_k(n)$ without Dedekind sums

A formula due to Selberg gives a faster (and remarkably **simple**) algorithm.

$$A_k(n) = \left(\frac{k}{3}\right)^{1/2} \sum_{(3\ell^2+\ell)/2 \equiv -n \bmod k} (-1)^\ell \cos\left(\frac{6\ell+1}{6k}\pi\right)$$

The index ranges over $0 \le \ell < 2k$ ($O(k^{1/2})$ terms are nonzero)

Brute force is efficient: testing whether $\ell$ solves the quadratic equation is much cheaper than evaluating a Dedekind sum

The cost for $p(n)$ is still $O(n)$ integer operations and $O(n^{3/4})$ cosines

## Evaluating $A_k(n)$ using prime factorization

Whiteman (1956): if $k = p^e$, then

$$A_k(n) = \sqrt{\frac{s}{t}} \cos\left(\frac{\pi r}{24k}\right)$$

where $r, s, t \in \mathbb{Z}$ are determined by certain quadratic modular equations, GCD and Jacobi symbol computations.

If $k = k_1 k_2$ where $\gcd(k_1, k_2) = 1$, then $A_k(n) = A_{k_1}(n_1) A_{k_2}(n_2)$ where $n_1$ and $n_2$ are solutions of modular equations.

**Algorithm**: factor $k$ into prime powers to write $A_k(n)$ as a product of $O(\log k)$ cosines.

## Cost of modular arithmetic

We can factor all $k$ in time $O(n^{1/2} \log n)$.

A single $k$ has at most $\log k$ prime factors

Jacobi symbol, multiplication, inverse, etc. mod $k$: $O(\log^{1+o(1)} k)$

Square roots mod $p$: $O(\log^{3+o(1)} p)$ using the Shanks-Tonelli algorithm, $O(\log^{2+o(1)} p)$ using Cipolla's algorithm (assuming the Extended Riemann Hypothesis!)

Cost of modular arithmetic for each $A_k(n)$: $O(\log^{3+o(1)} k)$ (assuming ERH)

Practical efficiency: optimized functions in FLINT!

## Fast high-precision numerical evaluation

We use MPFR functions (automatically giving asymptotically fast numerical evaluation), plus some improvements:

Compute $\pi$ using Hanhong Xue's `gmp-chudnovsky` program

For small $k$, compute $\exp(C/k)$ as $(\exp(C))^{1/k}$

For small $q$, compute $\cos(p\pi/q)$ using numerical Newton iteration to solve $P(\cos(2\pi/n)) = 0$ where $P(x)$ is an integer polynomial of degree $\phi(n)/2$.

$P$ can be generated numerically (balanced product) or symbolically (repeated decomposition into Chebyshev polynomials). Alternative: use $x^n - 1$ (complex arithmetic).

## Faster low-precision numerical evaluation

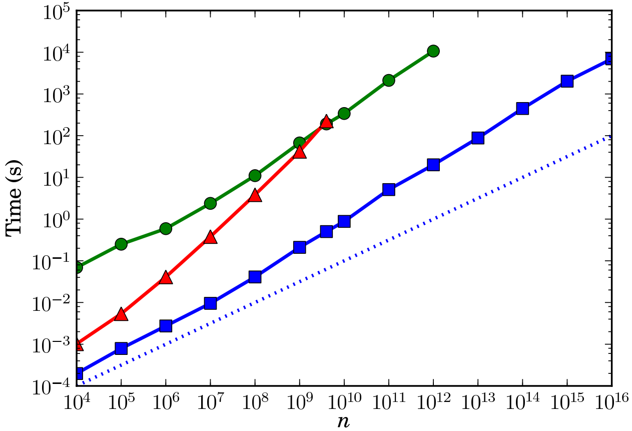
Most of the terms in the HRR sum are small (close to unit magnitude)

Use hardware double precision arithmetic when the needed precision falls below 53 bits

We need to make assumptions about the accuracy of system transcendental functions. Range reducing $x = p\pi/q$ to $(0, \pi/4)$ avoids potential catastrophic error in trigonometric functions.

Bober's implementation also uses quad-double (not used in FLINT)

## Computing $p(n)$ with FLINT, Mathematica, Sage



FLINT (blue squares), Mathematica 7 (green circles), Sage 4.7 (red triangles). Dotted line: $t = 10^{-6} n^{1/2}$.

## Timings

| $n$ | Mathematica 7 | Sage 4.7 | FLINT | First term |
|---|---|---|---|---|
| $10^4$ | 69 ms | 1 ms | 0.20 ms | |
| $10^5$ | 250 ms | 5.4 ms | 0.80 ms | |
| $10^6$ | 590 ms | 41 ms | 2.74 ms | |
| $10^7$ | 2.4 s | 0.38 s | 0.010 s | |
| $10^8$ | 11 s | 3.8 s | 0.041 s | |
| $10^9$ | 67 s | 42 s | 0.21 s | 43% |
| $10^{10}$ | 340 s | | 0.88 s | 53% |
| $10^{11}$ | 2,116 s | | 5.1 s | 48% |
| $10^{12}$ | 10,660 s | | 20 s | 49% |
| $10^{13}$ | | | 88 s | 48% |
| $10^{14}$ | | | 448 s | 47% |
| $10^{15}$ | | | 2,024 s | 39% |
| $10^{16}$ | | | 6,941 s | 45% |
| $10^{17}$ | | | 27,196* s | 33% |
| $10^{18}$ | | | 87,223* s | 38% |
| $10^{19}$ | | | 350,172* s | 39% |

## Near-optimality (in practice)

Computing the first term (basically $\pi$ and a single $\exp(x)$) takes nearly half the total running time

$\Rightarrow$ Improving the tail of the HRR sum further can give at most a twofold speedup

$\Rightarrow$ Parallelization of the HRR sum can give at most a twofold speedup

For a larger speedup, we would need faster high-precision transcendental functions (for example, using parallelization at the level of computing exp, or even in single multiplications)

## Large values of $p(n)$

| $n$ | Decimal expansion | Num. digits | Terms | Error |
|---|---|---|---|---|
| $10^{12}$ | 6129000962 . . . 6867626906 | 1,113,996 | 264,526 | $10^{-7}$ |
| $10^{13}$ | 5714414687 . . . 4630811575 | 3,522,791 | 787,010 | $10^{-8}$ |
| $10^{14}$ | 2750960597 . . . 5564896497 | 11,140,072 | 2,350,465 | $10^{-8}$ |
| $10^{15}$ | 1365537729 . . . 3764670692 | 35,228,031 | 7,043,140 | $10^{-9}$ |
| $10^{16}$ | 9129131390 . . . 3100706231 | 111,400,846 | 21,166,305 | $10^{-9}$ |
| $10^{17}$ | 8291300791 . . . 3197824756 | 352,280,442 | 63,775,038 | $10^{-9}$ |
| $10^{18}$ | 1478700310 . . . 1701612189 | 1,114,008,610 | 192,605,341 | $10^{-10}$ |
| $10^{19}$ | 5646928403 . . . 3674631046 | 3,522,804,578 | 582,909,398 | $10^{-11}$ |

The number of partitions of ten quintillion:
$p(10^{19}) = p(10000000000000000000) \approx 5.65 \times 10^{3,522,804,577}$

3.5 GB output, 97 CPU hours, $\sim$ 150 GB memory

## Combinatorial interpretations

$p(10^{15})$: the number of different ways the United States national debt ($\approx \$10^{13}$) can be paid off in bags of cents

$p(10^{19})$: the number of ways to form a collection of sticks whose lengths are multiples of 1 m, such that they add up to the thickness of the Milky Way galaxy ($\approx 10^{19}$ m) when placed end to end

## Finding congruences

Ramanujan (1919):

$$p(5k+4) \equiv 0 \pmod{5}$$
$$p(7k+5) \equiv 0 \pmod{7}$$
$$p(11k+6) \equiv 0 \pmod{11}$$

Ono (2000): for every prime $m \geq 5$, there exist infinitely many congruences of the type

$$p(Ak+B) \equiv 0 \bmod m$$

A constructive, computational procedure for finding such $(A, B, m)$ with $13 \leq m \leq 31$ was discovered by Weaver (2001)

## Theorem (Weaver)

Let $m \in \{13, 17, 19, 23, 29, 31\}$, $\ell \geq 5$ a prime, $\varepsilon \in \{-1, 0, 1\}$. If $(m, \ell, \varepsilon)$ satisfies a certain property, then $(A, B, m)$ is a partition function congruence where

$$A = m\ell^{4-|\varepsilon|}$$
$$B = \frac{m\ell^{3-|\varepsilon|}\alpha + 1}{24} + m\ell^{3-|\varepsilon|}\delta,$$

where $\alpha$ is the unique solution of $m\ell^{3-|\varepsilon|}\alpha \equiv -1 \bmod 24$ with $1 \leq \alpha < 24$, and where $0 \leq \delta < \ell$ is any solution of

$$\begin{cases} 24\delta \not\equiv -\alpha \bmod \ell & \text{if } \varepsilon = 0 \\ (24\delta + \alpha \mid \ell) = \varepsilon & \text{if } \varepsilon = \pm 1. \end{cases}$$

The free choice of $\delta$ gives $\ell - 1$ distinct congruences for a given tuple $(m, \ell, \varepsilon)$ if $\varepsilon = 0$, and $(\ell - 1)/2$ congruences if $\varepsilon = \pm 1$.

## Weaver's algorithm

```
Input: A pair of prime numbers $13 \le m \le 31$ and $\ell \ge 5$, $m \ne \ell$
Output: $(m, \ell, \varepsilon)$ defining a congruence, or Not-a-congruence
  $\delta_m \leftarrow 24^{-1} \bmod m$ {Reduced to $0 \le \delta_m < m$}
  $r_m \leftarrow (-m) \bmod 24$ {Reduced to $0 \le m < 24$}
  $v \leftarrow \frac{m-3}{2}$
  $x \leftarrow p(\delta_m)$ {We have $x \not\equiv 0 \bmod m$}
  $y \leftarrow p\left(m\left(\frac{r_m(\ell^2-1)}{24}\right) + \delta_m\right)$
  $f \leftarrow (3 \mid \ell)\left((-1)^v r_m \mid \ell\right)$ {Jacobi symbols}
  $t \leftarrow y + fx\ell^{v-1}$
  if $t \equiv \omega \bmod m$ where $\omega \in \{-1, 0, 1\}$ then
    return $(m, \ell, \omega\,(3(-1)^v \mid \ell))$
  else
    return Not-a-congruence
  end if
```

## Weaver's table

Weaver gives 76,065 congruences (167 tuples), obtained from a table of all $p(n)$ with $n < 7.5 \times 10^6$ (computed using the recursive Euler algorithm).

Limit on $\ell \approx 10^3$

Example: $m = 31$

$\varepsilon = 0$: $\ell = 107, 229, 283, 383, 463$

$\varepsilon \neq 0$: $(\ell, \varepsilon) = (101, 1), (179, 1), (181, 1), (193, 1), (239, 1), (271, 1)$

## New table

Testing all $\ell < 10^6$ resulted in 22 billion new congruences (70,359 tuples).

This involved evaluating $p(n)$ for $6(\pi(10^6) - 3) = 470,970$ distinct $n$, in parallel on $\approx 40$ cores (Hilbert computer at University of Warwick, courtesy of Bill Hart).

| $m$ | $\varepsilon = 0$ | $\varepsilon = +1$ | $\varepsilon = -1$ | Congruences | CPU | Max $n$ |
|---|---|---|---|---|---|---|
| 13 | 6,189 | 6,000 | 6,132 | 5,857,728,831 | 448 h | $5.9 \times 10^{12}$ |
| 17 | 4,611 | 4,611 | 4,615 | 4,443,031,844 | 391 h | $4.9 \times 10^{12}$ |
| 19 | 4,114 | 4,153 | 4,152 | 3,966,125,921 | 370 h | $3.9 \times 10^{12}$ |
| 23 | 3,354 | 3,342 | 3,461 | 3,241,703,585 | 125 h | $9.5 \times 10^{11}$ |
| 29 | 2,680 | 2,777 | 2,734 | 2,629,279,740 | 1,155 h | $2.2 \times 10^{13}$ |
| 31 | 2,428 | 2,484 | 2,522 | 2,336,738,093 | 972 h | $2.1 \times 10^{13}$ |
| All | 23,376 | 23,367 | 23,616 | 22,474,608,014 | 3,461 h | |

## Examples of new congruences

Example 1: $(13, 3797, -1)$ with $\delta = 2588$ gives

$$p(711647853449k + 485138482133) \equiv 0 \bmod 13$$

which we easily evaluate for all $k \leq 100$.

Example 2: $(29, 999959, 0)$ with $\delta = 999958$ gives

$$p(28995244292486005245947069k + 28995221336976431135321047)$$

$$\equiv 0 \bmod 29$$

This is out of reach for explicit evaluation ($n \approx 10^{25}$)

## Download the data

http://www.risc.jku.at/people/fjohanss/partitions/

or

http://sage.math.washington.edu/home/fredrik/partitions/

## Comparison of algorithms for vector computation

| $n$ | Series ($\mathbb{Z}/13\mathbb{Z}$) | Series ($\mathbb{Z}$) | HRR (all) | HRR (sparse) |
|---|---|---|---|---|
| $10^4$ | 0.01 s | 0.1 s | 1.4 s | 0.001 s |
| $10^5$ | 0.13 s | 4.1 s | 41 s | 0.008 s |
| $10^6$ | 1.4 s | 183 s | 1430 s | 0.08 s |
| $10^7$ | 14 s | | | 0.7 s |
| $10^8$ | 173 s | | | 8 s |
| $10^9$ | 2507 s | | | 85 s |

HRR competitive over $\mathbb{Z}$: when $n/c$ values are needed (our improvement: $c \approx 10$ vs $c \approx 1000$)

HRR competitive over $\mathbb{Z}/m\mathbb{Z}$: when $O(n^{1/2})$ values are needed (speedup for Weaver's algorithm: 1-2 orders of magnitude).

Most important advantages: little memory, parallel, resumable

## Conclusions

We can compute $p(n)$ with near-optimal complexity in both theory in practice.

Attention to asymptotics and implementation details allowed three order of magnitude improvement over previous implementations.

Properly implemented, the Hardy-Ramanujan-Rademacher formula turns out to be competitive with power series methods for mass evaluation of $p(n)$.

## The end

Thank you!