Efficient implementation of the Hardy-Ramanujan-Rademacher formula
or: Partitions in the quintillions

Fredrik Johansson
RISC-Linz

July 10, 2013

2013 SIAM Annual Meeting
San Diego, CA

Supported by Austrian Science Fund (FWF) grant Y464-N18
The partition function

\( p(n) \) counts the number of ways \( n \) can be written as the sum of positive integers without regard to order.

Example: \( p(4) = 5 \) since

\[
(4) = (3 + 1) = (2 + 2) = (2 + 1 + 1) = (1 + 1 + 1 + 1)
\]

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

\[ p(10) = 42 \]
\[ p(100) = 190569292 \]
\[ p(1000) = 24061467864032622473692149727991 \approx 2.4 \times 10^{31} \]
\[ p(10000) \approx 3.6 \times 10^{106} \]
\[ p(100000) \approx 2.7 \times 10^{346} \]
\[ p(1000000) \approx 1.5 \times 10^{1107} \]

\[ p(n) \sim \frac{1}{4n\sqrt{3}} e^{\pi \sqrt{2n/3}} \]

$p(n)$ has $\sim n^{1/2}$ digits
Euler’s method to compute $p(n)$

Generating function (Euler, 1748):

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

Recursive formula:

$$
p(n) = \sum_{k=1}^{n} (-1)^{k+1} \left( p \left( n - \frac{k(3k-1)}{2} \right) + p \left( n - \frac{k(3k+1)}{2} \right) \right)
$$

**Complexity:** $O(n^{3/2})$ integer operations, $O(n^2)$ bit operations
Asymptotically fast vector computation

Use fast power series arithmetic to expand

\[ \frac{1}{f(x)} = p(0) + p(1)x + \ldots + p(n)x^n + O(x^{n+1}) \]

The complexity is quasi-optimal for computing \( p(0), \ldots, p(n) \) simultaneously:

- \( O(n^{3/2+o(1)}) \) bit operations over \( \mathbb{Z} \)
- \( O(n^{1+o(1)}) \) bit operations over \( \mathbb{Z}/m\mathbb{Z} \) for fixed \( m \)

Calkin et al (2007): computation of \( p(n) \) mod \( m \) for all \( n \leq 10^9 \) and primes \( m \leq 103 \)
The Hardy-Ramanujan-Rademacher formula

There is a better way to compute an isolated value of $p(n)$, due to Hardy and Ramanujan (1917), Rademacher (1936):

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

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

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

Explicit error bound by Rademacher: can truncate after $O(n^{1/2})$ terms such that the error is smaller than $1/2$
How fast can we compute $p(n)$ using the HRR formula?

1938: Lehmer manually computes $p(599)$, $p(721)$

1995: Odlyzko claims that $p(n)$ can be computed in quasi-optimal time, but does not give a proof or an algorithm.

A few years ago:

- Implementations in several computer algebra systems: Pari/GP, Maple, Mathematica, Sage, etc. There are large differences in performance. Many versions give wrong values.
- No algorithmic analysis or implementation studies in the literature
- Largest reported values: $p(n)$, $n \approx 10^9$
Proof that \( p(n) \) can be computed in quasi-optimal time

A new implementation, running up to \( \sim 500 \) times faster than previous software (open source, part of FLINT, http://flintlib.org)

Error bounds for the main numerical parts of the algorithm

Discussion of implementation issues and practical optimizations

Large-scale \( p(n) \) computation, including generation of congruences
Theorem

\[ p(n) \text{ can be computed using } O(n^{1/2} \log^{4+o(1)} n) = O(n^{1/2+o(1)}) \]

bit operations.

This is quasi-optimal since \( p(n) \) has \( \Theta(n^{1/2}) \) bits.

- Unlike many sequences for which quasi-optimal algorithms are known, \( p(n) \) is not P-finite (holonomic)
- Quasi-optimal algorithms are not known for e.g. isolated Bell numbers (set partitions)
Cost of numerical evaluation

\[ p(n) = \sum_{k=0}^{N} T_k + \varepsilon \quad N = O(n^{1/2}), \quad \log_2 |T_k| = O(n^{1/2}/k) \]

Total area: \( O(n^{1/2} \log n) \)

We can compute \( p(n) \) in quasi-optimal time, if we can approximate \( T_k \) in quasi-optimal time.
Numerical evaluation of elementary functions

\[ T_k = (A_k(n) : \text{sum of roots of unity}) \times (\text{hyperbolic function}) \]

All numerical evaluation can be reduced to \textbf{elementary functions:}

- exp
- log
- sin
- sinh
- \ldots

Elementary functions can be evaluated to \( b \)-bit accuracy in quasi-optimal time \( O(b^{1+o(1)}) \).
Evaluating exponential sums

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

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

Naively:

- \( O(k^2) \) (integer/elementary function) operations for \( A_k(n) \)
- \( O(n^{3/2}) \) total (integer/elementary function) operations for \( p(n) \)

We need to get the cost for \( A_k(n) \) down to \( O(\log^c k) \) (integer/elementary function) 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

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

Fraction-free version by 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 $p(n)$

Still not good enough!
Evaluating $A_k(n)$ using prime factorization

Whiteman (1956):

- If $k = p^e$, then

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

- If $k = k_1 k_2$, $\gcd(k_1, k_2) = 1$, then

$$A_k(n) = A_{k_1}(n_1) A_{k_2}(n_2)$$

$r, s, t, n_1, n_2 \in \mathbb{Z}$ are determined by equations involving modular square roots, GCDs, Jacobi symbols, case distinctions.

**Algorithm**: factor $k$ into prime powers to write $A_k(n)$ as a product of $O(\log k)$ cosines. **Now the numerical evaluation becomes fast enough!**
Cost of integer arithmetic

**Factoring**: we do not know how to factor $k$ in $O(\log^c k)$ time. However, we can factor $1, \ldots, n^{1/2}$ simultaneously in time $O(n^{1/2} \log n)$.

**Integer arithmetic**: multiplication, GCD, \ldots: $O(\log^{1+o(1)} k)$

**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
- Must know a quadratic nonresidue mod $p$ (by a result of Erdős, a table for all $p < n^{1/2}$ can be precomputed sufficiently quickly)

**Total cost** of integer operations for $A_k(n)$: $O(\log^{3+o(1)} k)$
New implementation

2011:
- Using FLINT (integers) + MPFR (arbitrary-precision floats)
- A priori floating-point error bounds for the body of the algorithm
- Many numerical “tricks” without complete error bounds
  - Fast algorithms for $\pi$, roots of unity, …
  - Using hardware double-precision arithmetic

2013:
- Using FLINT + MPFR + Arb (new ball arithmetic library)
- “Tricks” reimplemented as proper Arb library functions, with proofs
- Code for $p(n)$ is simpler, with complete error bounds
Timings for $p(n)$ (2011)

Mathematica 7 (green circles)
Sage 4.7 (red triangles)
FLINT (blue squares)
# Timings for $p(n)$ (2011)

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

<table>
<thead>
<tr>
<th>$n$</th>
<th>Decimal expansion</th>
<th>Num. digits</th>
<th>Terms</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{12}$</td>
<td>61290000962...6867626906</td>
<td>1,113,996</td>
<td>264,526</td>
<td>$10^{-7}$</td>
</tr>
<tr>
<td>$10^{13}$</td>
<td>5714414687...4630811575</td>
<td>3,522,791</td>
<td>787,010</td>
<td>$10^{-8}$</td>
</tr>
<tr>
<td>$10^{14}$</td>
<td>2750960597...5564896497</td>
<td>11,140,072</td>
<td>2,350,465</td>
<td>$10^{-8}$</td>
</tr>
<tr>
<td>$10^{15}$</td>
<td>1365537729...3764670692</td>
<td>35,228,031</td>
<td>7,043,140</td>
<td>$10^{-9}$</td>
</tr>
<tr>
<td>$10^{16}$</td>
<td>9129131390...3100706231</td>
<td>111,400,846</td>
<td>21,166,305</td>
<td>$10^{-9}$</td>
</tr>
<tr>
<td>$10^{17}$</td>
<td>8291300791...3197824756</td>
<td>352,280,442</td>
<td>63,775,038</td>
<td>$10^{-9}$</td>
</tr>
<tr>
<td>$10^{18}$</td>
<td>1478700310...1701612189</td>
<td>1,114,008,610</td>
<td>192,605,341</td>
<td>$10^{-10}$</td>
</tr>
<tr>
<td>$10^{19}$</td>
<td>5646928403...3674631046</td>
<td>3,522,804,578</td>
<td>582,909,398</td>
<td>$10^{-11}$</td>
</tr>
</tbody>
</table>

The number of partitions of ten quintillion:

$p(10^{19}) = p(10000000000000000000000000) \approx 5.65 \times 10^{3,522,804,577}$

3.5 GB output, 97 CPU hours, $\sim$ 150 GB memory
New timings (2013, on slightly faster hardware)

<table>
<thead>
<tr>
<th>$n$</th>
<th>Mathematica 8.0</th>
<th>FLINT*</th>
<th>Arb**</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^6$</td>
<td>0.328 s</td>
<td>0.00147 s</td>
<td>0.00478 s</td>
</tr>
<tr>
<td>$10^9$</td>
<td>23.7 s</td>
<td>0.142 s</td>
<td>0.181 s</td>
</tr>
<tr>
<td>$10^{12}$</td>
<td>2458 s</td>
<td>11.32 s</td>
<td>11.50 s</td>
</tr>
<tr>
<td>$10^{15}$</td>
<td>307810 s</td>
<td>1109 s</td>
<td>1097 s</td>
</tr>
<tr>
<td>$10^{18}$</td>
<td>66738 s</td>
<td>57102 s</td>
<td></td>
</tr>
</tbody>
</table>

* 2011 implementation: using MPFR + hardware doubles (with incomplete error bounds)
** 2013 implementation: using ball arithmetic throughout to provably determine $p(n)$
Partition function congruences

Ramanujan (1919): for all \( k \in \mathbb{N} \),

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

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

\[
p(Ak + B) \equiv 0 \pmod{m}
\]
Algorithm to generate congruences (Weaver, 2001)

Defining tuple: 

- \( m \in \{13, 17, 19, 23, 29, 31\} \)
- \( \ell \geq 5 \) prime
- \( \varepsilon \in \{-1, 0, 1\} \)

For certain \( X, Y, Z \) where \( X = O(\ell^2) \), check the single case

\[
p(X) \equiv Y \mod Z
\]

If true, we obtain explicit \( A, B \) of size \( O(\ell^4) \) such that for all \( k \),

\[
p(Ak + B) \equiv 0 \mod m
\]

For a given tuple \((m, \ell, \varepsilon)\), there are \( O(\ell) \) such pairs \( A, B \), enumerated by an additional parameter \( \delta \).
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: $m = 31$

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

$\varepsilon \neq 0$: $(\ell, \varepsilon) = (101, 1), (179, 1), (181, 1), (193, 1), (239, 1), (271, 1)$
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 (hardware at University of Warwick, courtesy of Bill Hart)

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

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

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

which we may easily confirm for $k \leq 100$ by evaluation.

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

$$p(28995244292486005245947069k + 28995221336976431135321047)$$

$$\equiv 0 \mod 29$$

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

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

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

- Isolated values of $p(n)$ can be computed fast, both in theory and in practice.
- The HRR formula allows performing computations that are impractical with power series methods.
- Care is required for both asymptotics and implementation details.
- Generalizations: other HRR-type series for special types of partitions (into distinct parts, etc.), and possibly other number-theoretical computations.