COMPUTATIONS OF THE PARTITION FUNCTION, $p(n)$

JIMENA DAVIS AND ELIZABETH PEREZ

1. Introduction

The partition function has been a subject of great interest for many number theorists for the past several years. Although it is based on fairly simple ideas on the surface, there are still many elementary questions unanswered. The work of the Indian mathematician, Srinivasa Ramanujan(c.1887), has laid a basis for those interested in the theory of partitions. In 1966, two computational number theorists, Thomas Parkin and Daniel Shanks, wrote a paper on the parity of the partition function and supported their results with extensive computation, using the recursive definition for the partition function. Since then, an in-depth numerical analysis of partitions has not provided evidence to support the many conjectures being made in this area of mathematics. Today, especially in the past few years, great strides have been made and new ideas have taken shape by Ken Ono, Scott Ahlgren, and other leaders in the field. There are two main purposes of this paper. The first is to provide the reader with an efficient $O(N \log_2^2 N)$ algorithm for inverting a power series and computing the partition function modulo a large prime. The second is to provide data supporting conjectures concerning how the partition function is distributed modulo M.

2. The Partition Function

A **partition** is a sequence of positive integers which breaks a positive integer, n, into parts, where order does not matter. For example, the partitions of 4 are:

4
3,1
2,2
2,1,1
1,1,1,1

The partition number of 4 will be denoted as $p(4) = 5$ because there are five ways to partition 4. These numbers grow fairly quickly. For example, already at $n = 18$, $p(n) = 385$. From even just these two values, it is easy to see that it would be convenient to have a generating function for partitions. Parkin and Shanks[5] used the following generating function in their analysis of partitions:

$$\sum_{n=0}^{\infty} p(n)x^n = \left(1 + \sum_{n=1}^{\infty} (-1)^n [x^{n(3n-1)/2} + x^{n(3n+1)/2}]\right)^{-1}.$$

Date: July 18, 2002.

This is equivalent to Euler's[2] generating function for partitions, given by:

$$\sum_{n=0}^{\infty} p(n)q^n = \prod_{n=1}^{\infty} \frac{1}{1-q^n} = 1 + q + 2q^2 + 3q^3 + 5q^4 + 7q^5 + \dots \tag{1}$$

Euler's[2] Pentagonal Number Theorem states:

$$\prod_{n=1}^{\infty}(1-q^n) = \sum_{n=-\infty}^{\infty} (-1)^n q^{(3n^2+n)/2}. \tag{2}$$

The generating function for partitions may be simplified to the following equation:

$$\sum_{n=0}^{\infty} p(n)q^n = \frac{1}{\sum\limits_{n=-\infty}^{\infty} (-1)^n q^{(3n^2+n)/2}} \tag{3}$$

We will use this equation in our power series inversion algorithm.

The ratio of the number of partition numbers in residue class, r, to the total number of partition numbers is described by Scott Ahlgren and Ken Ono[1] as $\delta(r, m, X)$, where

$$\delta(r, m, X) = \frac{\#\{0 \leq n < X : p(n) \equiv r \pmod m\}}{X} \tag{4}$$

Examining this ratio for all residue classes modulo m gives us a feel for the distribution of the partition function among the residue classes of m. An important question is what happens to $\delta(r, m, X)$ as $X \to +\infty$. More specifically, does this limit even exist? If so, what is it?

We will investigate the following conjectures using the data collected:

Conjecture 2.1. *(Parkin and Shanks*[5]*)*

$$\lim_{X\to+\infty} \delta(0, 2, X) = \lim_{X\to+\infty} \delta(1, 2, X) = \frac{1}{2}$$

Conjecture 2.2. *(Ahlgren and Ono*[1]*)*

$$\lim_{X\to+\infty} \delta(0, 3, X) = \lim_{X\to+\infty} \delta(1, 3, X) = \lim_{X\to+\infty} \delta(2, 3, X) = \frac{1}{3}$$

Conjecture 2.3. *(Ahlgren and Ono*[1]*) For $l \geq 5$ and prime,*

$$\lim_{X\to+\infty} \delta(0, l, X) > \frac{1}{l}$$

If the previous conjecture is true, then the following question arises. Is it possible to predict $F(l)$ where,

$$F(l) = \lim_{X\to+\infty} \delta(0, l, X) - \frac{1}{l}? \tag{5}$$

Conjecture 2.4. *(Newman[4]) If M is a positive integer, then, for every integer r, where $0 \leq r < M$, there are infinitely many non-negative integers, n, for which $p(n) \equiv r \pmod{M}$.*

After we have established an algorithm for inverting a power series for the partition function, we will address these questions and show computational evidence for them.

3. Fast Fourier Tranforms

Before explaining our inversion algorithm, we must review some details about the Fast Fourier Transform(FFT) multiplication algorithm found in [3], a $O(N \log_2 N)$ algorithm for multiplying polynomials of size N. Begin by recognizing three symmetries in the roots of unity. Note that N is the total number of roots of unity, ω is the root of unity, and p is the prime modulus.

$$\omega^{i+N/2} \equiv p - \omega^i \text{ for } 0 \leq i \leq \frac{N}{2} - 1$$
$$(\omega^i)^{-1} \equiv p - \omega^{\frac{N}{2}-i} \text{ for } 0 \leq i \leq \frac{N}{2} - 1$$
$$(\omega^i)^{-1} \equiv \omega^{N-i} \text{ for } \frac{N}{2} \leq i \leq N - 1$$

These congruences allow us to calculate only $\frac{N}{2} + 1$ of the roots of unity rather than N of them. Additionally, we note that as the size of FFT halves, so does the number of roots of unity that are needed. FFT jumps through the roots appropriately, depending on its size.

After the roots of unity have been calculated, FFT is performed iteratively using bit reversal rather than recursively. To find a more in-depth discussion of Fast Fourier Transforms, see Walker[7].

4. Power Series Algorithm

Calculating partition numbers is slightly complicated because the generating function for partitions, (3), is an inverted power series. Therefore, we use an algorithm that provides an efficient way of inverting a power series, and hence, finding the partition numbers, using a Fast Fourier Transform multiplication algorithm found in [3] for polynomial multiplications. We also use GNU's big integer package, GMP, to allow for large partition numbers. This program may also be parallelized. In our case, we used the Message Passing Interface, MPI.

Given some integer, k, the algorithm will compute the partition numbers modulo a given prime, $p \equiv 1 \pmod{2^{k+1}}$, from 0 to $2^k - 1$. The user also inputs the prime's primitive root of unity, α. The basic idea behind the algorithm is to turn inversion into polynomial multiplications. We note the following:

Let $H_0(x) = h(x)$ be the polynomial to be inverted.
Also, $H_{i+1}(x^2) = H_i(x)H_i(-x)$.
Then,

$$\begin{aligned}
\frac{1}{h(x)} &= h(-x) \cdot \frac{1}{h(x)h(-x)} \\
&= H_0(-x) \cdot \frac{1}{H_1(x^2)} \\
&= H_0(-x) \cdot H_1(-x^2) \cdot \frac{1}{H_1(x^2)H_1(-x^2)} \\
&= H_0(-x) \cdot H_1(-x^2) \cdot H_2(-x^4) \cdot \frac{1}{H_2(-x^4)H_2(x^4)} \\
&\vdots \\
&= H_0(-x) \cdot H_1(-x^2) \cdot H_2(-x^4) \dots H_{k-1}(-x^{2^{k-1}}) \cdot \frac{1}{H_k x^{2^k}} \\
&= H_0(-x) \cdot H_1(-x^2) \cdot H_2(-x^4) \dots H_{k-1}(-x^{2^{k-1}})(1 + O(x^{2^k}))
\end{aligned}$$

Notice that the last term, $O(x^{2^k})$, is an error term so that the inversion is correct up to the $2^k - 1$ term, or, in our case, correct up to $p(2^k - 1)$. Therefore, to find all partition numbers correctly up to $p(2^k - 1)$, we must calculate H_i for $0 \leq i < k$ and then multiply all the H_i's together as mentioned above.

First of all, two main arrays of the size 2^{k+3} are kept. One holds the polynomials to be multiplied together, H_i's, and the other holds their Fast Fourier transforms, $\hat{H}_i$'s. The program uses the given primitive root, α, to find the first root of unity, $\omega = \alpha^{\frac{p-1}{2^{k+1}}} \pmod{p}$. This root of unity, ω, is then used to find the first $2^k + 1$ powers of ω, beginning with $\omega^0 = 1$. These will be stored in a third array and used in the Fast Fourier Transforms.

Next, we find the H_i's in the above equation. H_0 is our original polynomial, or the Pentagonal Numbers from (2). We determine how many terms of the Pentagonal Numbers equation 2 are needed for accuracy using the bound, b.

$$b = \frac{\sqrt{24 \cdot 2^{k+1} + 1} + 1}{6}$$

The rest of the H_i's are calculated in a loop beginning with $H_i(x) = H_0(x)$. The loop begins with a Fast Fourier Transform on $H_i(x)$ where the result is called $\hat{H}_i$. This transform is then point-wise multiplied with itself to obtain $\hat{H}_{i+1}(\hat{H}_{i+1}(j) = \hat{H}_i(j) \cdot \hat{H}_i(j + 2^{k+1})$. $\hat{H}_{i+1}$ is then multiplied through by the inverse of the size of the FFT. An inverse FFT is then performed to take $\hat{H}_{i+1}(x)$ to $H_{i+1}(x^2)$. In order to find this polynomial in terms of x, we collapse $H_{i+1}(x^2)$ by removing the odd terms(whose values are 0) and the result is $H_{i+1}(x)$. For accuracy, we only need half of this polynomial so we also truncate it before storing. This loop continues where the size of FFT is halved at the beginning of each round. The loop ends when the size of the FFT is 2. At the end of this step, we have all $H_i(x)$'s in one array and all $\hat{H}_i(x)$'s in another.

The next step in the inversion algorithm is to multiply all the H_i's together so that our final results are the partition numbers. This part of the algorithm is also

performed in a loop, where the FFTs grow in size. Let $G_0(x) = H_{k-1}(x)$. Also, $G_l(x) = H_{k-l-1}(x) \cdot G_{l-1}(-x^2)$ for $0 \leq l \leq k-1$. The loop begins by expanding and negating $G_i(x)$, resulting in $G_i(-x^2)$. Then, perform FFT to obtain $\hat{G}_i(-x^2)$. This polynomial is then point-wise multiplied with $\hat{H}_{k-2-i}(x)$ which is already stored in an array. The result is $\hat{G}_{i+1}(x)$. FFT is performed on this polynomial to obtain $G_{i+1}(x)$. For accuracy, only half of this polynomial is needed so we truncate it to conserve space. The size of the FFT then doubles and the process repeats until the size of the FFT reaches 2^{k+2}. The polynomial, $G_{k-1}(x)$, has negative odd terms. $G_{k-1}(-x)$ is found, which holds the partition numbers modulo the given prime, p.

Because this algorithm only finds the partition numbers modulo a prime, p, the inversion algorithm must be performed for many different primes. Then, the Chinese Remainder Theorem must be implemented to find the actual values of the partition numbers. Recall that the Chinese Remainder theorem says that for $(m, n) = 1$ and:

$$\begin{aligned} u &\equiv a \pmod{m} \\ u &\equiv b \pmod{n} \end{aligned}$$

there is a unique solution for u modulo $m \cdot n$. More specifically, $u \equiv any + bmx \pmod{mn}$. The Chinese Remainder Theorem is used to find the partition numbers modulo the product of all the primes used. If the product of the primes is large enough, then the partition numbers modulo the product is equivalent to the actual partition numbers. Finally, the process is complete and the partition numbers are found.

To see the power series inversion program in more detail, refer to Appendix A. To see the Chinese Remainder Program, refer to Appendix B.

5. Run-Time Analysis

Now that the algorithm has been described in detail, it is necessary to analyze its run-time. To run through the inversion algorithm with one prime, it takes $O(N \log_2^2 N)$ multiplications, where N is 2^{k+2} and also the size of the largest FFT performed. There are two main parts of the algorithm; one where the H_i's are calculated and the other where they are multiplied together. Both parts iterate through a loop $\log_2 N$ times. The number of multiplications in the two main parts differ by a constant so we will only analyze one loop. Also, all the FFTs in the program are of size N or smaller. This means that each FFT performs $O(N \log_2 N)$ multiplications(see Chiu[3]). To simplify things, we will assume all FFTs are of size N to obtain an upper bound on the number of multiplications in the program. Each loop contains two FFTs [each having $O(N \log_2 N)$ multiplications], a point-wise multiplication [$O(N)$ multiplications], and multiplication by N^{-1}[$O(N)$ multiplications]. When added together, we obtain the number of multiplications for one loop. As mentioned before, there are $\log_2 N$ iterations of one loop and two main parts. The upper bound for the total number of multiplications, $M(N)$, is:

$$\begin{aligned} M(N) &= 2 \log_2 N [O(N \log_2 N) + O(N) + O(N)] \\ &= O(N \log_2^2 N) \end{aligned}$$

$M(N)$ is the number of multiplications used to find the partitions numbers modulo one prime. The algorithm must be repeated for enough primes so that the actual partition number values may be calculated. To find the number of primes, we use an upper bound of the Hardy-Ramanujan asymptotic formula, which was perfected by Rademacher[2]:

$$p(n) \sim \frac{1}{4n\sqrt{3}} \cdot e^{\pi\sqrt{\frac{2n}{3}}} < e^{\pi\sqrt{\frac{2n}{3}}}$$

Assuming the primes are chosen to be of size 2^{61}, then the inversion algorithm must be performed $\log_{2^{61}} e^{\pi\sqrt{\frac{2n}{3}}}$ times. To do this, an upper bound of $\log_{2^{61}} e^{\pi\sqrt{\frac{2n}{3}}} \cdot M(N)$ multiplications is required. .

6. Computational Evidence

The most exciting part of this program is the data it can produce. As mentioned earlier in this paper, many conjectures have been made regarding partitions. Using our program, the user can not only find the partition numbers for large integers, but they may also find congruences within the partition function.

Consider first Conjecture 2.1 concerning the partition function modulo 2. It is expected that about half of the partition numbers are odd and half of them are even. We have not yet produced data on this conjecture but expect to be able to before our final paper.

Now, recall Conjecture 2.2 concerning the distribution of the partition function modulo 3. Data is available which allows us to test the conjecture, shown in Figure 1. All data was calculated using the first 4194304 partition numbers. The percentage of partition numbers that are congruent to 0 modulo 3 remains around 33.3% with slight fluctuations depending on how many partition numbers are included in the calculations. The percentage of partition numbers congruent to 1 and 2 behaves similarly. This data tells us that Conjecture 2.2 is a strong one, assuming that the limits mentioned in it exist.

Figure 1. Percentages for Residue Classes Modulo 3

The following table shows the slight variation of the percentages modulo 3 as more partition numbers were used. The final line in the table holds the data included in Figure 1.

# of Partition Numbers	0 mod 3	1 mod 3	2 mod 3
16	0.312500	0.250000	0.437500
507920	0.334370	0.332068	0.333562
1015824	0.333036	0.333693	0.333271
2015248	0.333069	0.333771	0.333160
3014672	0.332821	0.333943	0.333237
4194304	0.333098	0.333635	0.333267

Similar data can be collected for other primes. We computed the residue class percentages for all primes from 5 to 101. See Appendix B for graphs representing some of these percentages for the first 4194304 partition numbers. This data also assists with the analysis of Conjecture 2.3, which states that the percentage of partition numbers congruent to 0 modulo a prime, $l \geq 5$, is greater than $\frac{1}{l}$. To see this more clearly on the included graphs, $y = \frac{1}{l}$ is also plotted. Additionally, $y = \frac{1}{l} \pm \frac{1}{l^2}$ is plotted to see the variation from $\frac{1}{l}$.$Thisalsoallowsustoview$F(l) graphically. Let's consider the distribution modulo 5. The following graph shows that $\delta(0, 5, 4194304) = 0.363980$, but $\frac{1}{5} = 0.20$, a significant difference. Other $\delta(r, 5, 4194304)$ for $0 < r < 5$ appear flat and less than $\frac{1}{5}$.

FIGURE 2. Percentages for Residue Classes Modulo 5

This result is expected because of the Ramanujan[6] congruence $p(5n + 4) \equiv 0 \pmod 5$. Similar results are also expected for 7 and 11 due to the Ramanujan congruences $p(7n + 5) \equiv 0 \pmod 7$ and $p(11n + 6) \equiv 0 \pmod{11}$. Therefore, the graphs representing the percentages for the primes $l = 7$ and 11 have a similar pattern to Figure 2.

Graphs for moduli greater than 11 begin to look a little different, including more variation about $\frac{1}{l}$ among the residue classes. Because of the scaling of our pictures, this variation is not extremely significant. However, considering Conjecture 2.3, it is useful to look at a graph for prime l where $\delta(0, l, 4194304) < \frac{1}{l}$. Such a result occurs at $l = 73$, shown in Figure 3 for the first 4194304 partition numbers. This result occurs for primes much less than 73 but l=73 seems most interesting right now. When we considered the first 245776 partition numbers, 1.3707% of them were

FIGURE 3. Percentages for Residue Classes Modulo 73

congruent to 0 (mod 73). However, 245776 is the last value of X for which our data shows $\delta(0, l, X) > \frac{1}{l}$. Note that Conjecture 2.3 refers to a limit as $X \to +\infty$ so the conjecture is not necessarily disproved. It is surprising, though, that the majority of our data for a modulus of 73 does not support Conjecture 2.3. Additionally, these results imply that $F(l)$(see Eq. (5)) may not be as useful to investigate.

Finally, consider Conjecture 2.4. We have not done a lot of analysis of data produced for this conjecture, but it seems to support it.

7. Acknowledgements

The authors thank the National Science Foundation for their generous support. Additionally, we would like to thank Dr. Kevin James, Dr. Neil Calkin, and Charles Swannack for their help.

References

[1] S. Ahlgren and K. Ono, *Congruences and conjectures for the partition function*, Contemporary Math. **291** (2001), 1-9.

[2] G.E. Andrews, *The Theory of Partitions*, Cambridge University Press, Cambridge, 1984.

[3] P. Chiu, *Transforms, Finite Fields, and Fast Multiplication*, Mathematics **63, no. 5** (Dec. 1990), 330-336.
[4] M. Newman, *Periodicity modulo m and divisibility properties of the partition function*, Trans. Amer. Math. Soc. **97** (1960), 225-236.
[5] T. Parkin and D. Shanks, *On the Distribution of Parity in the Partition Function*, Math. of Computation **21, Issue 99** (Jul 1967), 466-480.
[6] S. Ramanujan, *Congruences properties of partitions*,Proceedings of the London Mathematical Society **19** (1919), 207-210.
[7] J. Walker, *Fast Fourier Transforms*, CRC, Boca Raton, FL, 1996.

Appendix A

Appendix B

Appendix C