Congruent Number Theta Coefficients to 10^{12}

William Hart, Gonzalo Tornaria, Mark Watkins

July 15, 2010

Congruent Numbers

- **Definition (Congruent Number)** An integer n is *congruent* if it is the area of a right triangle with rational length sides.

Congruent Numbers

- **Definition (Congruent Number)** An integer n is *congruent* if it is the area of a right triangle with rational length sides.
- E.g. $5, 6, 7, 13, 14, 15, 20, 21, 22, 23, 24, 28, \ldots.$

Congruent Numbers

- **Definition (Congruent Number)** An integer n is *congruent* if it is the area of a right triangle with rational length sides.
- E.g. $5, 6, 7, 13, 14, 15, 20, 21, 22, 23, 24, 28, \ldots.$
- 5 is the area of the $20/3, 3/2, 41/6$ triangle.

Congruent Numbers

- **Definition (Congruent Number)** An integer n is *congruent* if it is the area of a right triangle with rational length sides.
- E.g. $5, 6, 7, 13, 14, 15, 20, 21, 22, 23, 24, 28, \ldots.$
- 5 is the area of the $20/3, 3/2, 41/6$ triangle.
- *Equivalently* n is congruent if there exist rational x, y, z, w such that

$$x^2 + ny^2 = z^2 \text{ and } x^2 - ny^2 = w^2.$$

Congruent Numbers

- **Definition (Congruent Number)** An integer n is *congruent* if it is the area of a right triangle with rational length sides.
- E.g. $5, 6, 7, 13, 14, 15, 20, 21, 22, 23, 24, 28, \ldots$.
- 5 is the area of the $20/3, 3/2, 41/6$ triangle.
- *Equivalently* n is congruent if there exist rational x, y, z, w such that

$$x^2 + ny^2 = z^2 \text{ and } x^2 - ny^2 = w^2.$$

- Congruent n correspond to points (u^2, v) on the elliptic curve $E_n : y^2 = x^3 - n^2x$.

Tunnel's Criterion

Theorem (Tunnell)

Let n be an odd squarefree positive integer. Set

$$a(n) = \#\{(x,y,z) \in \mathbb{Z}^3 \mid x^2 + 2y^2 + 8z^2 = n\} - 2\,\#\{(x,y,z) \in \mathbb{Z}^3 \mid x^2 + 2y^2 + 32z^2 = n\},$$

$$b(n) = \#\{(x,y,z) \in \mathbb{Z}^3 \mid x^2 + 4y^2 + 8z^2 = n\} - 2\,\#\{(x,y,z) \in \mathbb{Z}^3 \mid x^2 + 4y^2 + 32z^2 = n\}.$$

Tunnel's Criterion

Theorem (Tunnell)

Let n be an odd squarefree positive integer. Set

$$\begin{aligned} a(n) = \#\{(x,y,z) \in \mathbb{Z}^3 \mid x^2 + 2y^2 + 8z^2 = n\} \\ - 2\,\#\{(x,y,z) \in \mathbb{Z}^3 \mid x^2 + 2y^2 + 32z^2 = n\}, \end{aligned}$$

$$\begin{aligned} b(n) = \#\{(x,y,z) \in \mathbb{Z}^3 \mid x^2 + 4y^2 + 8z^2 = n\} \\ - 2\,\#\{(x,y,z) \in \mathbb{Z}^3 \mid x^2 + 4y^2 + 32z^2 = n\}. \end{aligned}$$

If n is congruent then $a(n) = 0$. If $2n$ is congruent then $b(n) = 0$.

Tunnel's Criterion

Theorem (Tunnell)

Let n be an odd squarefree positive integer. Set

$$a(n) = \#\{(x,y,z) \in \mathbb{Z}^3 \mid x^2 + 2y^2 + 8z^2 = n\} \\ - 2\,\#\{(x,y,z) \in \mathbb{Z}^3 \mid x^2 + 2y^2 + 32z^2 = n\},$$

$$b(n) = \#\{(x,y,z) \in \mathbb{Z}^3 \mid x^2 + 4y^2 + 8z^2 = n\} \\ - 2\,\#\{(x,y,z) \in \mathbb{Z}^3 \mid x^2 + 4y^2 + 32z^2 = n\}.$$

If n is congruent then $a(n) = 0$. If $2n$ is congruent then $b(n) = 0$. Moreover, if the weak BSD conjecture is true for the curve $y^2 = x^3 - n^2 x$ then the converses also hold: $a(n) = 0$ implies n is congruent and $b(n) = 0$ implies $2n$ is congruent.

Theta functions

- Define $\theta_t = \sum_{m=-\infty}^{\infty} q^{tm^2}$.

Theta functions

- Define $\theta_t = \sum_{m=-\infty}^{\infty} q^{tm^2}$.
-

$$\theta_8(\theta_1 - \theta_4) \times (\theta_8 - 2\theta_{32}) = \sum_{n\equiv 1 \pmod 8} a(n)\, q^n,$$

$$(\theta_2 - \theta_8)(\theta_1 - \theta_4) \times (\theta_8 - 2\theta_{32}) = \sum_{n\equiv 3 \pmod 8} a(n)\, q^n,$$

$$\theta_{16}(\theta_1 - \theta_4) \times (\theta_8 - 2\theta_{32}) = \sum_{n\equiv 1 \pmod 8} b(n)\, q^n,$$

$$(\theta_4 - \theta_{16})(\theta_1 - \theta_4) \times (\theta_8 - 2\theta_{32}) = \sum_{n\equiv 5 \pmod 8} b(n)\, q^n.$$

Cyclic Convolution

Definition (Convolution) Given two vectors of length n

$$A = [a_0, a_1, \ldots, a_{n-1}]$$

and

$$B = [b_0, b_1, \ldots, b_{n-1}]$$

the *cyclic convolution* of A, B is

$$C = [c_0, c_1, \ldots, c_{n-1}]$$

where

$$c_k = \sum_{i+j \equiv k \pmod{n}} a_i b_j$$

.

Polynomial Multiplication

Given polynomials of length n,

$$f_1(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$$

$$f_2(x) = b_0 + b_1 x + \cdots + b_{n-1} x^{n-1}$$

computing the product polynomial

$$f_1 f_2(x) = c_0 + c_1 x + \cdots c_{2n-2} x^{2n-2}$$

is *linear* or *acyclic convolution*.

$$c_k = \sum_{i+j=k} a_i b_j.$$

Convolution

- Cyclic convolution is polynomial multiplication mod $x^n - 1$.

Convolution

- Cyclic convolution is polynomial multiplication mod $x^n - 1$.
- Linear convolution (polynomial multiplication) can be performed by zero padding to length $2n$

$$A = [a_0, a_1, \ldots, a_{n-1}, 0, 0, \ldots, 0]$$

$$B = [b_0, b_1, \ldots, b_{n-1}, 0, 0, \ldots, 0]$$

then perform cyclic convolution (polynomial multiplication modulo $x^{2n} - 1$).

Negacyclic Convolution

- The *negacyclic convolution* is polynomial multiplication modulo $x^n + 1$.

Negacyclic Convolution

- The *negacyclic convolution* is polynomial multiplication modulo $x^n + 1$.
- Can be computed by performing the transformation $x \mapsto \zeta_n y$ with ζ_n a primitive $2n$-th root of unity ($\zeta_n^n = -1$).

Negacyclic Convolution

- The *negacyclic convolution* is polynomial multiplication modulo $x^n + 1$.
- Can be computed by performing the transformation $x \mapsto \zeta_n y$ with ζ_n a primitive $2n$-th root of unity ($\zeta_n^n = -1$).
- Now perform multiplication modulo $y^n - 1$ using cyclic convolution.

FFT Trick

- To compute multiplication modulo $x^{2n} - 1$, compute it modulo $x^n - 1$ using the cyclic convolution and compute it modulo $x^n + 1$ using the negacyclic convolution, then recombine using CRT

FFT Trick

- To compute multiplication modulo $x^{2n} - 1$, compute it modulo $x^n - 1$ using the cyclic convolution and compute it modulo $x^n + 1$ using the negacyclic convolution, then recombine using CRT
- The CRT step is an addition, a subtraction and division by 2 (called rescaling)

FFT Trick

- To compute multiplication modulo $x^{2n} - 1$, compute it modulo $x^n - 1$ using the cyclic convolution and compute it modulo $x^n + 1$ using the negacyclic convolution, then recombine using CRT
- The CRT step is an addition, a subtraction and division by 2 (called rescaling)
- If $n = 2^k d$ is divisible by a power of 2, can iterate the FFT trick a further k times

The DIF FFT

- Reduction mod $x^n - 1$ and $x^n + 1$ combined with the negacyclic transformation $x \mapsto \zeta_n y$ is called a Decimation In Frequency (DIF) FFT butterfly

The DIF FFT

- Reduction mod $x^n - 1$ and $x^n + 1$ combined with the negacyclic transformation $x \mapsto \zeta_n y$ is called a Decimation In Frequency (DIF) FFT butterfly
- $A = [s_0, s_1, \ldots, s_{n-1}, t_0, t_1, \ldots, t_{n-1}]$

The DIF FFT

- Reduction mod $x^n - 1$ and $x^n + 1$ combined with the negacyclic transformation $x \mapsto \zeta_n y$ is called a Decimation In Frequency (DIF) FFT butterfly
- $A = [s_0, s_1, \ldots, s_{n-1}, t_0, t_1, \ldots, t_{n-1}]$
- DIF_FFT_butterfly(A) =

$$[s_0 + t_0, s_1 + t_1, \ldots, s_{n-1} + t_{n-1},$$

$$s_0 - t_0, \zeta_n(s_1 - t_1), \ldots, \zeta_n^{n-1}(s_{n-1} - t_{n-1})]$$

The DIF FFT

- Reduction mod $x^n - 1$ and $x^n + 1$ combined with the negacyclic transformation $x \mapsto \zeta_n y$ is called a Decimation In Frequency (DIF) FFT butterfly
- $A = [s_0, s_1, \ldots, s_{n-1}, t_0, t_1, \ldots, t_{n-1}]$
- DIF_FFT_butterfly$(A) =$

$$[s_0 + t_0, s_1 + t_1, \ldots, s_{n-1} + t_{n-1},$$

$$s_0 - t_0, \zeta_n(s_1 - t_1), \ldots, \zeta_n^{n-1}(s_{n-1} - t_{n-1})]$$

- DIF FFT applies a DIF butterfly then squares the root of unity ζ_n and recurses first on the left half, then on the right half

The Inverse FFT

- Reversing the negacyclic transformation $y \mapsto \zeta_n^{-1}x$ followed by CRT recombination (without rescaling) is called an inverse FFT (IFFT) butterfly

The Inverse FFT

- Reversing the negacyclic transformation $y \mapsto \zeta_n^{-1} x$ followed by CRT recombination (without rescaling) is called an inverse FFT (IFFT) butterfly
- $A = [s_0, s_1, \ldots, s_{n-1}, t_0, t_1, \ldots, t_{n-1}]$

The Inverse FFT

- Reversing the negacyclic transformation $y \mapsto \zeta_n^{-1} x$ followed by CRT recombination (without rescaling) is called an inverse FFT (IFFT) butterfly
- $A = [s_0, s_1, \ldots, s_{n-1}, t_0, t_1, \ldots, t_{n-1}]$
- DIF_IFFT_butterfly(A) =

$$[s_0 + t_0, s_1 + \zeta_n^{-1} t_1, \ldots, s_{n-1} + \zeta_n^{1-n} t_{n-1},$$

$$s_0 - t_0, s_1 - \zeta_n^{-1} t_1, \ldots, s_{n-1} - \zeta_n^{1-n} t_{n-1}]$$

The Inverse FFT

- Reversing the negacyclic transformation $y \mapsto \zeta_n^{-1} x$ followed by CRT recombination (without rescaling) is called an inverse FFT (IFFT) butterfly
- $A = [s_0, s_1, \ldots, s_{n-1}, t_0, t_1, \ldots, t_{n-1}]$
- DIF_IFFT_butterfly$(A) =$

$$[s_0 + t_0, s_1 + \zeta_n^{-1} t_1, \ldots, s_{n-1} + \zeta_n^{1-n} t_{n-1},$$
$$s_0 - t_0, s_1 - \zeta_n^{-1} t_1, \ldots, s_{n-1} - \zeta_n^{1-n} t_{n-1}]$$

- DIF IFFT recurses first on the left half, then on the right half, then applies a DIF IFFT butterfly

The FFT Convolution

To compute the cyclic convolution using the FFT and IFFT, suppose we have two vectors of length $n = 2^k d$

The FFT Convolution

To compute the cyclic convolution using the FFT and IFFT, suppose we have two vectors of length $n = 2^k d$

- Compute k levels of FFT butterflies

The FFT Convolution

To compute the cyclic convolution using the FFT and IFFT, suppose we have two vectors of length $n = 2^k d$

- Compute k levels of FFT butterflies
- Perform multiplications mod $x^d - 1$, called "pointwise multiplications"

The FFT Convolution

To compute the cyclic convolution using the FFT and IFFT, suppose we have two vectors of length $n = 2^k d$

- Compute k levels of FFT butterflies
- Perform multiplications mod $x^d - 1$, called "pointwise multiplications"
- Perform k levels of IFFT butterflies

The FFT Convolution

To compute the cyclic convolution using the FFT and IFFT, suppose we have two vectors of length $n = 2^k d$

- Compute k levels of FFT butterflies
- Perform multiplications mod $x^d - 1$, called "pointwise multiplications"
- Perform k levels of IFFT butterflies
- Rescale by 2^k

The FFT Convolution

To compute the cyclic convolution using the FFT and IFFT, suppose we have two vectors of length $n = 2^k d$

- Compute k levels of FFT butterflies
- Perform multiplications mod $x^d - 1$, called "pointwise multiplications"
- Perform k levels of IFFT butterflies
- Rescale by 2^k
- If $n = 2^k$ FFT convolution can be performed in time $O(n \log n)$ coefficient operations

Polynomials over Z/pZ

- Break $f(x)$ in $R = \mathbb{Z}/p\mathbb{Z}[x]$ into pieces of length 2^{k-1} and zero pad each to length 2^k

Polynomials over Z/pZ

- Break $f(x)$ in $R = \mathbb{Z}/p\mathbb{Z}[x]$ into pieces of length 2^{k-1} and zero pad each to length 2^k
- Consider these to be coefficients of a polynomial

Polynomials over Z/pZ

- Break $f(x)$ in $R = \mathbb{Z}/p\mathbb{Z}[x]$ into pieces of length 2^{k-1} and zero pad each to length 2^k
- Consider these to be coefficients of a polynomial
- Think of these “coefficients” as being in the ring $S = R/(x^{2^k} + 1)$

Polynomials over Z/pZ

- Break $f(x)$ in $R = \mathbb{Z}/p\mathbb{Z}[x]$ into pieces of length 2^{k-1} and zero pad each to length 2^k
- Consider these to be coefficients of a polynomial
- Think of these “coefficients” as being in the ring $S = R/(x^{2^k} + 1)$
- Perform an FFT over the ring S

Polynomials over Z/pZ

- Break $f(x)$ in $R = \mathbb{Z}/p\mathbb{Z}[x]$ into pieces of length 2^{k-1} and zero pad each to length 2^k
- Consider these to be coefficients of a polynomial
- Think of these "coefficients" as being in the ring $S = R/(x^{2^k} + 1)$
- Perform an FFT over the ring S
- Note x is a 2^{k+1}-th root of unity

Polynomials over Z/pZ

- Break $f(x)$ in $R = \mathbb{Z}/p\mathbb{Z}[x]$ into pieces of length 2^{k-1} and zero pad each to length 2^k
- Consider these to be coefficients of a polynomial
- Think of these "coefficients" as being in the ring $S = R/(x^{2^k} + 1)$
- Perform an FFT over the ring S
- Note x is a 2^{k+1}-th root of unity
- Can use the negacyclic transformation to do pointwise multiplications in S, or algorithm of your choice

Cache Friendliness

- The DIF FFT is fairly cache friendly as the working set quickly becomes localised

Cache Friendliness

- The DIF FFT is fairly cache friendly as the working set quickly becomes localised
- The Matrix Fourier Algorithm improves cache locality

Cache Friendliness

- The DIF FFT is fairly cache friendly as the working set quickly becomes localised
- The Matrix Fourier Algorithm improves cache locality
- Write $y = x^{2^{n_1}}$ and let

$$f(x) = f_1(y) + xf_2(y) + x^2 f_3(y) + \cdots + x^{2^{n_1}-1} f_{2^{n_1}-1}(y)$$

with each f_i of length 2^{n_2}

Cache Friendliness

- The DIF FFT is fairly cache friendly as the working set quickly becomes localised
- The Matrix Fourier Algorithm improves cache locality
- Write $y = x^{2^{n_1}}$ and let

$$f(x) = f_1(y) + xf_2(y) + x^2 f_3(y) + \cdots + x^{2^{n_1}-1} f_{2^{n_1}-1}(y)$$

  with each f_i of length 2^{n_2}
- Can be thought of as:

Cache Friendliness

- The DIF FFT is fairly cache friendly as the working set quickly becomes localised
- The Matrix Fourier Algorithm improves cache locality
- Write $y = x^{2^{n_1}}$ and let

  $$f(x) = f_1(y) + xf_2(y) + x^2 f_3(y) + \cdots + x^{2^{n_1}-1} f_{2^{n_1}-1}(y)$$

  with each f_i of length 2^{n_2}
- Can be thought of as:
  (i) do 2^{n_1} FFT's of length 2^{n_2} (reduce the coeffs of the f_i's mod $y - \zeta^i$)

Cache Friendliness

- The DIF FFT is fairly cache friendly as the working set quickly becomes localised
- The Matrix Fourier Algorithm improves cache locality
- Write $y = x^{2^{n_1}}$ and let

$$f(x) = f_1(y) + xf_2(y) + x^2 f_3(y) + \cdots + x^{2^{n_1}-1} f_{2^{n_1}-1}(y)$$

  with each f_i of length 2^{n_2}
- Can be thought of as:
  (i) do 2^{n_1} FFT's of length 2^{n_2} (reduce the coeffs of the f_i's mod $y - \zeta^i$)
  (ii) twist by powers of ζ (reduce the x^j's mod the appropriate $x - \zeta^i$'s)

Cache Friendliness

- The DIF FFT is fairly cache friendly as the working set quickly becomes localised
- The Matrix Fourier Algorithm improves cache locality
- Write $y = x^{2^{n_1}}$ and let

$$f(x) = f_1(y) + xf_2(y) + x^2 f_3(y) + \cdots + x^{2^{n_1}-1} f_{2^{n_1}-1}(y)$$

  with each f_i of length 2^{n_2}
- Can be thought of as:
  (i) do 2^{n_1} FFT's of length 2^{n_2} (reduce the coeffs of the f_i's mod $y - \zeta^i$)
  (ii) twist by powers of ζ (reduce the x^j's mod the appropriate $x - \zeta^i$'s)
  (iii) do 2^{n_2} FFT's of length 2^{n_1} (reduce mod each $x - \zeta^i$)

Out-of-core FFT

- Can do length 2^{n_2} and 2^{n_1} FFT's in memory and keep rest of data on disk using Matrix Fourier Algorithm

Out-of-core FFT

- Can do length 2^{n_2} and 2^{n_1} FFT's in memory and keep rest of data on disk using Matrix Fourier Algorithm
- Problem: power series truncation can only be done after all FFT's complete

Out-of-core FFT

- Can do length 2^{n_2} and 2^{n_1} FFT's in memory and keep rest of data on disk using Matrix Fourier Algorithm
- Problem: power series truncation can only be done after all FFT's complete
- Solution:

Out-of-core FFT

- Can do length 2^{n_2} and 2^{n_1} FFT's in memory and keep rest of data on disk using Matrix Fourier Algorithm
- Problem: power series truncation can only be done after all FFT's complete
- Solution:
  (i) bundle blocks of coefficients using Kronecker Segmentation into large integer coefficients

Out-of-core FFT

- Can do length 2^{n_2} and 2^{n_1} FFT's in memory and keep rest of data on disk using Matrix Fourier Algorithm
- Problem: power series truncation can only be done after all FFT's complete
- Solution:
  (i) bundle blocks of coefficients using Kronecker Segmentation into large integer coefficients
  (ii) Reduce large coefficients modulo many small primes

Out-of-core FFT

- Can do length 2^{n_2} and 2^{n_1} FFT's in memory and keep rest of data on disk using Matrix Fourier Algorithm
- Problem: power series truncation can only be done after all FFT's complete
- Solution:
  (i) bundle blocks of coefficients using Kronecker Segmentation into large integer coefficients
  (ii) Reduce large coefficients modulo many small primes
  (iii) Perform power series multiplications over $\mathbb{Z}/p\mathbb{Z}$ in-core, performing truncation in-core

Out-of-core FFT

- Can do length 2^{n_2} and 2^{n_1} FFT's in memory and keep rest of data on disk using Matrix Fourier Algorithm
- Problem: power series truncation can only be done after all FFT's complete
- Solution:
  (i) bundle blocks of coefficients using Kronecker Segmentation into large integer coefficients
  (ii) Reduce large coefficients modulo many small primes
  (iii) Perform power series multiplications over $\mathbb{Z}/p\mathbb{Z}$ in-core, performing truncation in-core
  (iv) Recombine using CRT

Out-of-core FFT

- Can do length 2^{n_2} and 2^{n_1} FFT's in memory and keep rest of data on disk using Matrix Fourier Algorithm
- Problem: power series truncation can only be done after all FFT's complete
- Solution:
  (i) bundle blocks of coefficients using Kronecker Segmentation into large integer coefficients
  (ii) Reduce large coefficients modulo many small primes
  (iii) Perform power series multiplications over $\mathbb{Z}/p\mathbb{Z}$ in-core, performing truncation in-core
  (iv) Recombine using CRT
  (v) Recover small coefficients from large product coefficients

Out-of-core FFT

- Can do length 2^{n_2} and 2^{n_1} FFT's in memory and keep rest of data on disk using Matrix Fourier Algorithm
- Problem: power series truncation can only be done after all FFT's complete
- Solution:
  (i) bundle blocks of coefficients using Kronecker Segmentation into large integer coefficients
  (ii) Reduce large coefficients modulo many small primes
  (iii) Perform power series multiplications over $\mathbb{Z}/p\mathbb{Z}$ in-core, performing truncation in-core
  (iv) Recombine using CRT
  (v) Recover small coefficients from large product coefficients
- Lose $\log n$ factor in complexity (due to CRT), but gain factor of 2 in I/O and disk space

Congruent numbers up to 10^{12}

- In 2009 we computed 10^{12} terms of the congruent number theta function by multiplying power series

Congruent numbers up to 10^{12}

- In 2009 we computed 10^{12} terms of the congruent number theta function by multiplying power series
- Used David Harvey's zn_poly (or FLINT) for the polynomial multiplication over $\mathbb{Z}/p\mathbb{Z}$

Congruent numbers up to 10^{12}

- In 2009 we computed 10^{12} terms of the congruent number theta function by multiplying power series
- Used David Harvey's zn_poly (or FLINT) for the polynomial multiplication over $\mathbb{Z}/p\mathbb{Z}$
- Wrote code for Kronecker segmentation, modular reduction, CRT, transposes, etc

Congruent numbers up to 10^{12}

- In 2009 we computed 10^{12} terms of the congruent number theta function by multiplying power series
- Used David Harvey's zn_poly (or FLINT) for the polynomial multiplication over $\mathbb{Z}/p\mathbb{Z}$
- Wrote code for Kronecker segmentation, modular reduction, CRT, transposes, etc
- Used OpenMP for parallelism (16 cores)

Congruent numbers up to 10^{12}

- In 2009 we computed 10^{12} terms of the congruent number theta function by multiplying power series
- Used David Harvey's zn_poly (or FLINT) for the polynomial multiplication over $\mathbb{Z}/p\mathbb{Z}$
- Wrote code for Kronecker segmentation, modular reduction, CRT, transposes, etc
- Used OpenMP for parallelism (16 cores)
- Used mmap kernel service for disk I/O

Congruent numbers up to 10^{12}

- In 2009 we computed 10^{12} terms of the congruent number theta function by multiplying power series
- Used David Harvey's zn_poly (or FLINT) for the polynomial multiplication over $\mathbb{Z}/p\mathbb{Z}$
- Wrote code for Kronecker segmentation, modular reduction, CRT, transposes, etc
- Used OpenMP for parallelism (16 cores)
- Used mmap kernel service for disk I/O
- Code now part of FLINT (thetaproduct.c)

1 (mod 8) Class

10^9	10^{10}	10^{11}
3801661	21768969	142778019

2×10^{11}	3×10^{11}	4×10^{11}
127475330	115249740	107930081

5×10^{11}	6×10^{11}	7×10^{11}
102774355	98817294	95656907

8×10^{11}	9×10^{11}	10^{12}
93030373	90748990	88803354

Table 1 : Congruent numbers in the 1 (mod 8) class.

3 (mod 8) Class

10^9	10^{10}	10^{11}
2921535	17019170	112979066

2×10^{11}	3×10^{11}	4×10^{11}
101436853	91949066	86213764

5×10^{11}	6×10^{11}	7×10^{11}
82196846	79106503	76626341

8×10^{11}	9×10^{11}	10^{12}
74546400	72781203	71239101

Table 2 : Congruent numbers in the $3 \pmod 8$ class.

2 (mod 16) Class

10^9	10^{10}	10^{11}
2110645	12294626	81759844

2×10^{11}	3×10^{11}	4×10^{11}
73445274	66579936	62455317

5×10^{11}	6×10^{11}	7×10^{11}
59536672	57282587	55504389

8×10^{11}	9×10^{11}	10^{12}
53993974	52728711	51619397

Table 3 : Congruent numbers in the $2 \pmod{16}$ class.

10 (mod 16) Class

10^9	10^{10}	10^{11}
1842072	10842882	72556705

2×10^{11}	3×10^{11}	4×10^{11}
65378932	59347550	55720114

5×10^{11}	6×10^{11}	7×10^{11}
53152609	51190025	49599296

8×10^{11}	9×10^{11}	10^{12}
48268971	47158661	46159584

Table 4 : Congruent numbers in the $10 \pmod{16}$ class.