Systematic Computations on Mertens' Conjecture and Dirichlet's Divisor Problem by Vectorized Sieving

Dedicated to Cor Baayen, at the occasion of his retirement as scientific director of SMC and its CWI

Walter M. Lioen
CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
and
Jan van de Lune
Noordermiedweg 31, 9074 LM Hallum, The Netherlands

In this paper we present two vectorized numerical sieve algorithms for the number theoretical functions $\mu(n)$ and $\tau(n)$. These sieve algorithms are generalizations of Eratosthenes' sieve for finding prime numbers. We show algorithms for fast systematic computations on Mertens' conjecture and Dirichlet's divisor problem. We have implemented the algorithm for Mertens' conjecture on a Cray C90 and performed a systematic computation of extremes of $M(x)/\sqrt{x}$ up to 10^{13}. We established the bounds $-0.513 < M(x)/\sqrt{x} < 0.571$, valid for $200 < x \leq 10^{13}$.

1 Introduction

Eratosthenes' sieve is one of the oldest algorithms in number theory (3rd century B.C.). The ultimate sieving device for Eratosthenes' sieve and its generalizations is a (parallel) vector computer or a massively parallel computer. Our generalizations of Eratosthenes' sieve are devised with large (parallel) vector computers in mind. They are virtually 100 percent vectorizable and they become more and more efficient when the amount of memory increases.

We start by introducing Mertens' conjecture in Section 2. Section 3 is devoted to a completely vectorized algorithm for a systematic computation of $M(x)$ and analysis of $M(x)/\sqrt{x}$. In Section 4 we describe Dirichlet's divisor problem. The corresponding algorithm is given in Section 5. This algorithm in its turn is a generalization of the algorithm described in Section 3. In Section 6 a selection of the numerical results for $M(x)/\sqrt{x}$, $x = 1, \ldots, 10^{13}$, is presented. Finally, in the last section we give some concluding remarks.

2 Mertens' Conjecture

The Möbius function $\mu(n)$ is defined as follows

$$\mu(n) = \begin{cases} 1, & n = 1, \\ 0, & \text{if } n \text{ is divisible by a prime square,} \\ (-1)^k, & \text{if } n \text{ is the product of } k \text{ distinct primes.} \end{cases}$$

We consider $M(x)$, the first summatory function of $\mu(n)$,

$$M(x) = \sum_{n \leq x} \mu(n).$$

$M(x)$ describes the difference between the number of squarefree positive integers $n \leq x$ with an even number of prime factors and those with an odd number of prime factors.

Based on a table of $M(x)$ for $x = 1, \ldots, 10000$ Mertens [11] conjectured that

$$|M(x)| < \sqrt{x}, \qquad x > 1.$$

Later, based on more extensive numerical 'evidence', Von Sterneck [17] even conjectured that

$$|M(x)| < \frac{1}{2}\sqrt{x}, \qquad x > 200.$$

The Möbius function is related to the Riemann zeta function by

$$\frac{1}{\zeta(s)} = \sum_{n=1}^{\infty} \frac{\mu(n)}{n^s}, \qquad \Re(s) > 1.$$

Boundedness of $M(x)/\sqrt{x}$ implies the truth of the Riemann hypothesis. However, the converse does not hold.

For the history of the function $M(x)/\sqrt{x}$ and the disproves of Von Sterneck's conjecture and later Mertens' conjecture—both first theoretical and later effective—we refer to [16]. A comprehensive bibliography may be found in the paper by Odlyzko and Te Riele [13] in which they disprove Mertens' conjecture.

Although it is known that $M(x)/x \to 0$ as $x \to \infty$ (and even more than this), the best known effective asymptotic upper bound on $|M(x)|$ to date [4] is

$$|M(x)| \leq \frac{1}{2360}x, \qquad x \geq 617973.$$

3 A vectorized algorithm for $M(x)/\sqrt{x}$

3.1 Eratosthenes' sieve

Eratosthenes indicated the following method of obtaining all the primes in the range $2, \ldots, N$: put all numbers between 2 and N into a 'sieve'; as long as the sieve is not empty, select the smallest number remaining in the sieve, and strike out all multiples of this prime number. The complexity of both the

sieve initialization and the prime number selection is $\mathcal{O}(N)$. The complexity of striking out all multiples of the prime numbers found and therewith the complexity of Eratosthenes' sieve is

$$\sum_{i=1}^{\pi(\sqrt{N})} \left\lfloor \frac{N}{p_i} \right\rfloor \sim N \log\log\sqrt{N},$$

where $\pi(x)$ denotes the number of prime numbers not exceeding x. Usually, one only sieves the odd numbers. Moreover, if N becomes large one has to partition the sieve interval. Even for N large, 10^{13}, say, $\log\log\sqrt{N}$ is fairly small. This gives an almost linear complexity $\mathcal{O}(N)$. For the sake of completeness: the best (sub)linear prime number sieve has complexity $\mathcal{O}(N/\log\log N)$, cf. [10, 15].

3.2 Sieving $\mu(n)$

The following algorithm yields the Möbius function $\mu(n)$ for $n = 1, \ldots, N$.

for $n = 1$ to N
    $\mu(n) = 1$
for all $p \leq \sqrt{N}$
    for all $n, p \mid n$
        $\mu(n) = -p \cdot \mu(n)$
for all $p \leq \sqrt{N}$
    for all $n, p^2 \mid n$
        $\mu(n) = 0$
for $n = 1$ to N
    if $|\mu(n)| \neq n$ then
        $\mu(n) = -\mu(n)$
for $n = 1$ to N
    $\mu(n) = sign(\mu(n))$

This algorithm starts initializing a sieve array μ with the value 1. Besides the sieve array we also keep a list of all primes not exceeding $\sqrt{N}$. Next, for all prime numbers p not greater than $\sqrt{N}$ we multiply $\mu(n)$ by $-p$ for every n a multiple of p. By multiplying with $-p$ we achieve two things: 1. we multiply by p in order to see if we end up having handled all prime factors of n; 2. by multiplying with $-p$ instead of p, we keep track of the parity of the number of different prime factors of n handled so far. For all prime numbers p not greater than $\sqrt{N}$ we set $\mu(n)$ to 0 for every n a multiple of p^2. After this step we check whether $|\mu(n)| = n$ holds. If $|\mu(n)| = n$ holds, n is squarefree and none of its prime factors is greater than $\sqrt{N}$. If $|\mu(n)| = n$ does not hold, we have two possibilities: either $\mu(n) = 0$, in which case n is not squarefree, or n is squarefree and has exactly one prime divisor $p > \sqrt{N}$. Anyhow, if $|\mu(n)| = n$ does not hold, we just change the sign of $\mu(n)$, taking care of the parity for this last prime factor, or a no-op if $\mu(n) = 0$. At this point we have three possibilities: 1. $\mu(n) = 0$, if n is not squarefree; 2. $\mu(n) < 0$, if n is the product

of an odd number of distinct primes; 3. $\mu(n) > 0$, if n is the product of an even number of distinct primes. With the obvious definition of *sign*, the last loop in the algorithm above completes the computation of the Möbius function $\mu(n)$ for $n = 1, \ldots, N$. It is easy to see that the complexity for the above algorithm is the same as for Eratosthenes' sieve: $\mathcal{O}(N \log\log \sqrt{N})$.

As we already mentioned for Eratosthenes' sieve, we have to partition the sieve interval if N becomes large. The determination of whether a prime hits a partition and, if so, the first index it hits, is a non-vectorizable process. In order not to loose vector speed one should choose the partition size considerably (10–100 times, say) larger than the number of sieve primes. In our computations N equals 10^{13}, so the number of sieve primes becomes $\pi(\sqrt{N}) = 227{,}647$. We chose our partition size equal to 10^7.

3.3 Small prime variation

Since we want to compute all values of $M(x)$ systematically, we can not halve the amount of work by only sieving the odd numbers, as we can for Eratosthenes' sieve. For the same reason we can not apply a 'small prime variation' as in MPQS [14]. However, it is possible to apply a different kind of small prime variation: replace the initialization $\mu(n) = 1$ by a 'block-initialization'. Using the small primes $2, 3, 5, 7, 11$, say, and also the small prime squares $4, 9$, we get a pattern-length of

$$2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 2 \cdot 3 = 13{,}860.$$

Sieving with only these few primes and prime squares requires

$$\left\lfloor \frac{N}{2} \right\rfloor + \left\lfloor \frac{N}{3} \right\rfloor + \left\lfloor \frac{N}{5} \right\rfloor + \left\lfloor \frac{N}{7} \right\rfloor + \left\lfloor \frac{N}{11} \right\rfloor + \left\lfloor \frac{N}{4} \right\rfloor + \left\lfloor \frac{N}{9} \right\rfloor \approx 1.6N$$

sieve updates. If we store the initial pattern of length 13,860 and do a periodic block initialization of the sieve array with this pattern (instead of the total initialization with 1), we get about $1.6N$ sieve updates, for these small primes and prime powers, for free.

We did not tell the full story by stating that one can not manage sieving only the odd numbers. As pointed out by Tijdeman [18] one may use the identity $\mu(n) + \mu(2n) = 0$, for n odd, together with the identity $\mu(4n) = 0$, to avoid the computation of $\mu(n)$ for n even. However, for N large, so that we have to partition the sieve array to get the job done, this becomes impractical because one would have to store some $N/4$ intermediate $\mu(n)$-values for n even.

3.4 Vectorizing the partial summation

Thus far we described a vectorized algorithm for the systematic computation of the Möbius function. Eventually, however, we are interested in the extremes of $M(x)/\sqrt{x}$, so that first of all we have to compute the partial sums

$$M(x) = \sum_{n \le x} \mu(n), \qquad x = 1, \ldots, N.$$

Phrasing this as an algorithm one might compute the partial sums as follows.

$M(1) = \mu(1)$
for $x = 2$ to N
    $M(x) = \mu(x) + M(x-1)$

The previous loop is a classical example of a non-vectorizable loop because of its recursion on $M(x-1)$.

Assuming that the array M initially contains the values of μ (we do this computation in-place anyhow), and partitioning the array in chunks of length s, $1 \leq s \leq N$ we can compute the partial sums using the following algorithm.

for $y = 2$ to s
    for $x = y$ to N by s
        $M(x) = M(x) + M(x-1)$

$l = \lfloor N/s \rfloor s$
for $y = s+1$ to l by s
    for $x = y$ to $y+s-1$
        $M(x) = M(x) + M(y-1)$

for $x = l+1$ to N
    $M(x) = M(x) + M(l)$

Here, the first loop nest solves $\lceil N/s \rceil$ independent partial summation problems. The inner loop of the first loop nest performs the same operation simultaneously on all chunks. Because of the increment s, this inner loop is not recursive, therefore vectorizable. After executing the first loop nest, the original partial summation problem is only solved for the first chunk $M(x), x = 1, \ldots, s$. The second loop nest takes care of the other chunks in turn by adding $M(y-1)$, the end-point-value of the previous chunk, to all values in the current chunk. Here, the inner loop is vectorizable, since trivially $y-1 < y, \ldots, y+s-1$. After executing the second loop nest, all chunks except for possibly the last one also contain the correct values for the original partial summation problem. Finally, the last loop handles the last chunk in case s does not evenly divide N.

Using this algorithm, also the partial summation is vectorizable albeit at the price of doing twice as many additions but, at a performance gain of an order of magnitude, because it now readily vectorizes. On a Cray C90 we measured a speed-up factor of 5–9 depending on the values of N and s.

We still have not chosen the chunk size s. In order to perform both loop nests at vector speed, s should be chosen such that the iteration counts of the respective inner loops (being N/s and s) are not too small. Moreover, s, being the increment of the first inner loop, should not be a multiple of the number of memory banks. The latter would cause memory bank conflicts resulting in a measured performance degradation by a factor 4 in CPU time on a Cray C90. Finally, the choice of s also depends on other optimization techniques in the actual implementation.

Had our prototype not been written using Fortran `INTEGER`s, we probably would have opted for Cray's SCILIB (SCIentific LIBrary) routine `RECPS`. Our implementation and `RECPS` perform comparably. In Section 5 we can not do without the partial summation algorithm described above, since there is no SCILIB routine with the same functionality for `INTEGER`s.

3.5 Gathering the statistics

Having a completely vectorized algorithm for the systematic computation of $M(x)$ we are still not completely done. One not entirely minor point remains: we want to study the local extremes of $M(x)/\sqrt{x}$. Clearly, we do not want—neither have to—compute $\sqrt{x}$ for all x. $M(x)/\sqrt{x}$ can only reach a new extreme value if $M(x)$ does. Searching for new extremes can only be done at vector speed if the number of extremes is small with respect to the number of elements we are considering. If in the interval we are investigating Mertens' conjecture

$$-\sqrt{x} < M(x) < \sqrt{x}, \qquad 1 < x \leq N,$$

holds, it guarantees at most $\sqrt{N}$ local maxima and minima. On the other hand, if Mertens' conjecture would not hold in the interval we are investigating, we would find the smallest argument value x giving a counterexample for Mertens' conjecture.

We search $M(x)$ for new extremes in either direction using the highly efficient Cray SCILIB routines `ISRCHFGT` and `ISRCHFLT`.

We refrain from describing the actual bookkeeping process, since bookkeeping of the extremes gets rather complicated by the sieve partitioning, our search for extremes in two directions (positive/negative), and minimization of the printed output.

3.6 Comparison to Neubauer's algorithm and Dress' version

Te Riele drew our attention to the work of Neubauer [12], who used a similar algorithm for computing $M(x)$, $x = 1, \ldots, 10^8$. Neubauer also had to partition his sieve interval. However, he used three sieve arrays. His algorithm [12, p. 2] reads as follows: for $n = 0, 1, \ldots$

$$1000n < m \leq 1000(n+1)$$

$$p_i^2 \mid m \Rightarrow \mu(m) = 0$$

$$\varrho(m) = \sum_{p_i \mid m} \log p_i \qquad \nu(m) = \sum_{p_i \mid m} 1$$

Neubauer builds up $\varrho(m)$ to check whether there is a prime factor $p > \sqrt{N}$ and he counts the number of different prime factors in $\nu(m)$. Neubauer does not use multiplications, nor divisions. However, he must take care of precision because of the inherently inexact $\log p_i$ values.

Recently, Dress used a variant of Neubauer's algorithm using only two sieve arrays (a and μ in [3, Algorithme 2]) and—at least in the description—a division step.

Our algorithm only uses one sieve array, computing $\mu(n)$ in-place. Moreover, on current vector computers a vector-multiplication is just as expensive as a vector-addition. Because of the overhead involved in partitioning the sieve interval, there is a certain trade-off between memory usage and CPU usage. Using only one sieve array, and, of course, the unavoidable prime table, it is possible to use the available memory as efficient as possible. Moreover, we have added a small prime variation.

4 Dirichlet's divisor problem

We consider $D(x)$, the first summatory function of $\tau(n)$,

$$D(x) = \sum_{n \le x} \tau(n),$$

where $\tau(n)$ denotes the number of divisors of n. Dirichlet [2] proved that

$$D(x) = x \log x + (2\gamma - 1)x + E(x),$$

where γ is Euler's constant, and $E(x) = \mathcal{O}(\sqrt{x})$. This may be considered as a lattice point problem, counting the number of lattice points in the first quadrant between the axes and the hyperbola $qd = x$, including those on the hyperbola

$$D(x) = \sum_{n \le x} \sum_{d \mid n} 1 = \sum_{\substack{q,d \\ qd \le x}} 1.$$

Compare FIGURE 1. An unsolved problem in analytic number theory is the estimation of the order of the error term $E(x)$. TABLE 1 shows the historical development of Dirichlet's divisor problem. For a more complete table, further references, and much more about lattice point problems in general, we refer to [5, 7].

5 A vectorized algorithm for Dirichlet's divisor problem

Given the unique prime factorization of n

$$n = \prod_{i=1}^{k} p_i^{e_i}$$

we have the following formula for $\tau(n)$

$$\tau(n) = \prod_{i=1}^{k} (e_i + 1).$$

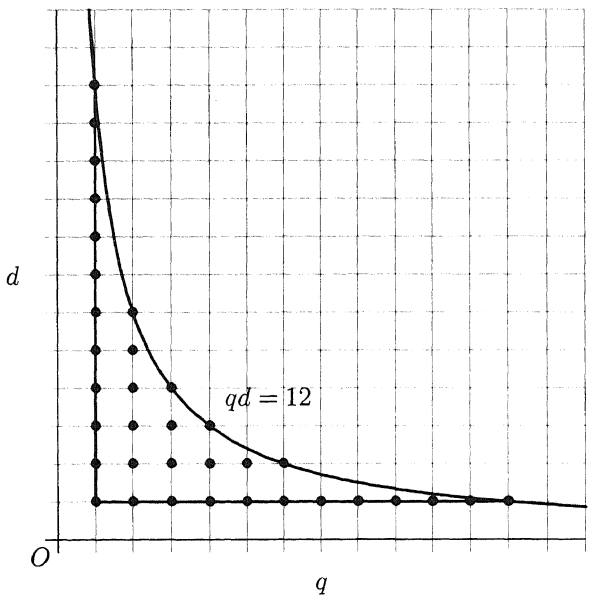

FIGURE 1. Dirichlet's divisor problem

Using two sieve arrays instead of one, and exploiting the above mentioned formula we can sieve $\tau(n)$ similarly as $\mu(n)$. The following algorithm computes the number of divisors function $\tau(n)$ for $n = 1, \ldots, N$.

for $n = 1$ to N
    $I(n) = 1$
    $\tau(n) = 1$
for all $p \leq \sqrt{N}$
    for all $n, p \mid n$
        $I(n) = p \cdot I(n)$
        $\tau(n) = 2 \cdot \tau(n)$
for $e = 2$ to $\lfloor \log_2 N \rfloor$
    for all $p \leq \sqrt[e]{N}$
        for all $n, p^e \mid n$
            $I(n) = p \cdot I(n)$
            $\tau(n) = (e+1) \cdot \dfrac{\tau(n)}{e}$
for $n = 1$ to N
    if $I(n) \neq n$ then
        $\tau(n) = 2 \cdot \tau(n)$

We need two sieve arrays because keeping track of a parity as for $\mu(n)$ does not suffice. In the I-array we multiply all prime factors encountered during sieving.

TABLE 1. The order of the error term in $D(x)$

	year	$E(x)$	
Dirichlet	1849	$\mathcal{O}(x^{1/2})$	
Voronoï	1903	$\mathcal{O}(x^{1/3}\log x)$	
Van der Corput	1922	$\mathcal{O}(x^{33/100})$	
Kolesnik	1969	$\mathcal{O}(x^{(12/37)+\varepsilon})$	$\forall \varepsilon > 0$
Iwaniec & Mozzochi [6]	1988	$\mathcal{O}(x^{7/22})$	
Van de Lune and Wattel conjecture [9]	1990	$\mathcal{O}(x^{1/4}\log x)$	
Hardy and Landau	1915	$\Omega_{\pm}(x^{1/4})$	

This way only a single prime factor $p > \sqrt{N}$ can remain which is taken care of by the last loop nest. In the τ-array we maintain the number of divisors using the above mentioned formula: when sieving with a prime factor we multiply $\tau(n)$ with 2 (since $e = 1$); sieving with a prime square we divide the current value of $\tau(n)$ by 2 and multiply with 3; when sieving with higher prime powers, exponent e, say, we divide by e and multiply with $e + 1$.

Similarly as for the $\mu(n)$ we can use a small prime variation by creating patterns for both the I-array and the τ-array.

The partial summation, and gathering of the statistics can all be performed analogous to the procedures for $M(x)/\sqrt{x}$.

For an actual implementation on the Cray C90 one should use an `INTEGER` τ-array, because of the very fast but inexact floating point division (resulting e.g. in $3.0/3.0 \neq 1$).

6 NUMERICAL RESULTS FOR $M(x)/\sqrt{x}$

We verified the results of Neubauer [12], Cohen & Dress [1], and Dress [3]. Furthermore, we established the bounds $-0.513 < M(x)/\sqrt{x} < 0.571$, valid for $200 < x \leq 10^{13}$. See TABLE 2 for some selected values of $M(x)$ and $M(x)/\sqrt{x}$ for $x = 1, \ldots, 10^{13}$.

The computation of Cohen and Dress [1] in 1979 up to $7.8 \cdot 10^9$ took a week on a TI980B minicomputer. The computation of Dress [3] up to 10^{12} in 1992 took 4000 hours on three Sun SPARCstations 2.

Our results were all obtained using one processor. A test run up to 10^{10} of our prototype implementation took 32 minutes on a Cray Y-MP. The same run of our final implementation took 9 minutes on a Cray C90. The speed-up was due to the faster machine and the improved implementation. The verification of [3] (up to 10^{12}) took some 17 hours on a Cray C90. Finally, the computation up to 10^{13} took a little less than 200 hours on a Cray C90.

TABLE 2. $M(x)$ and $M(x)/\sqrt{x}$ for some selected[a] $x < 10^{13}$

x	$M(x)$	$\frac{M(x)}{\sqrt{x}}$	x	$M(x)$	$\frac{M(x)}{\sqrt{x}}$
30,095,923	−1,448	−0.264	9,826,066,363	−31,207	−0.315
30,919,091	−2,573	−0.463	15,578,669,387	−51,116	−0.410
34,750,986	1,420	0.241	18,835,808,417	50,287	0.366
61,913,863	2,845	0.362	19,890,188,718	60,442	0.429
70,497,103	−2,574	−0.307	22,745,271,553	−51,117	−0.339
76,015,339	−3,448	−0.395	38,066,335,279	−81,220	−0.416
90,702,782	2,846	0.299	48,201,938,615	60,443	0.275
92,418,127	3,290	0.342	48,638,777,062	76,946	0.349
109,528,655	−3,449	−0.330	56,794,153,135	−81,221	−0.341
110,103,729	−4,610	−0.439	101,246,135,617	−129,332	−0.406
141,244,329	3,291	0.277	106,512,264,731	76,947	0.236
152,353,222	4,279	0.347	108,924,543,546	170,358	0.516
179,545,614	−4,611	−0.344	148,449,169,741	−129,333	−0.336
179,919,749	−6,226	−0.464	217,309,283,735	−190,936	−0.410
216,794,087	4,280	0.291	295,766,642,409	170,359	0.313
360,718,458	6,695	0.353	297,193,839,495	207,478	0.381
455,297,339	−6,227	−0.292	325,813,026,298	−190,937	−0.335
456,877,618	−8,565	−0.401	330,138,494,149	−271,317	−0.472
514,440,542	6,696	0.295	330,486,258,610[c]	−287,440	−0.500
903,087,703	10,246	0.341	330,508,686,218[c]	−294,816	−0.513
1,029,223,105	−8,566	−0.267	400,005,203,086	207,479	0.328
1,109,331,447	−15,335	−0.460	661,066,575,037	331,302	0.407
1,228,644,631	10,247	0.292	1,246,597,697,210	−294,817	−0.264
2,218,670,635	15,182	0.322	1,440,355,022,306	−368,527	−0.307
2,586,387,614	−15,336	−0.302	1,600,597,184,945	331,303	0.262
2,597,217,086	−17,334	−0.340	1,653,435,193,541	546,666	0.425
3,061,169,989	15,183	0.274	2,008,701,330,005	−368,528	−0.260
3,314,385,678	21,777	0.378	2,087,416,003,490	−625,681	−0.433
3,724,183,273	−17,335	−0.284	2,319,251,110,865	546,667	0.359
3,773,166,681	−25,071	−0.408	2,343,412,610,499	594,442	0.388
5,439,294,226	21,778	0.295	3,268,855,616,262	−625,682	−0.346
5,439,294,781	21,791	0.295	3,270,926,424,607	−635,558	−0.351
6,600,456,626	−25,072	−0.309	3,754,810,967,055	594,443	0.307
6,631,245,058	−31,206	−0.383	4,098,484,181,477	780,932	0.386
7,544,459,107	21,792	0.251	5,184,088,665,413	−635,559	−0.279
7,660,684,541	38,317	0.438	5,197,159,385,733	−689,688	−0.303
7,725,038,629[b]	43,947	0.500	6,202,507,744,370	780,933	0.314
7,766,842,813[b]	50,286	0.571	9,784,334,467,058	889,948	0.285

[a] A listed $M(x)$-value guarantees the corresponding x to be the smallest argument value for which $M(x)$ assumes this value. Consecutive $M(x)$-column-entries of the same sign guarantees absence of new extremal $M(x)$-values of the opposite sign in between. A framed $M(x)/\sqrt{x}$ value guarantees the corresponding x to be the smallest argument value greater than 200 for which $M(x)/\sqrt{x}$ assumes this value.

[b] This verifies a result of Cohen and Dress [1].

[c] This verifies a result of Dress [3].

7 Concluding remarks

We showed two vectorized algorithms: one for fast systematic computations on Mertens' conjecture, and one for fast systematic computations on Dirichlet's divisor problem. In an update of this paper we will extend Section 6 with numerical results for Dirichlet's divisor problem.

The algorithms we described are generalizable to arbitrary arithmetical functions $f : \mathbf{N} \to \mathbf{Z}$ as long as we have a fairly simple relation between $f(p^e q)$ and $f(p^{e-1}q)$, where $e, p, q \in \mathbf{N}$, p prime, $p \nmid q$. For example $\tau(p^e q) = \frac{e+1}{e}\tau(p^{e-1}q)$. In particular, we have devised similar algorithms for Gauß' lattice point problem and amicable numbers, to name just two [8].

Acknowledgements

This work was sponsored by the Stichting Nationale Computerfaciliteiten (National Computing Facilities Foundation, NCF) for the use of supercomputer facilities, with financial support from the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (Netherlands Organization for Scientific Research, NWO).

References

1. H. Cohen. Arithmétique et informatique. *Astérisque*, 61:57–61, 1979.
2. G.L. Dirichlet. Über die Bestimmung der mittleren Werthe in der Zahlentheorie. *Abhandlungen der Königlich Preussischen Akademie der Wissenschaften*, pages 69–83, 1849.
3. F. Dress. Fonction sommatoire de la fonction de Möbius, 1. Majorations expérimentales. *Experiment. Math.*, 2(2):89–98, 1993.
4. F. Dress and M. El Marraki. Fonction sommatoire de la fonction de Möbius, 2. Majorations asymptotiques élémentaires. *Experiment. Math.*, 2(2):99–112, 1993.
5. F. Fricker. *Einführung in die Gitterpunktlehre.* Number 73 in LMW/MA. Birkhäuser Verlag, 1982.
6. H. Iwaniec and C.J. Mozzochi. On the divisor and circle problems. *J. Number Theory*, 29(1):60–93, 1988.
7. E. Krätzel. *Lattice Points.* Number 22 in Mathematische Monographien. VEB Deutscher Verlag der Wissenschaften, 1988.
8. W.M. Lioen and J. van de Lune. Vectorized algorithms for certain arithmetical functions. Work in progress, 1995.
9. J. van de Lune and E. Wattel. Systematic computations on Dirichlet's divisor problem. To appear.
10. H.G. Mairson. Some new upper bounds on the generation of prime numbers. *Comm. ACM*, 20(9):664–669, September 1977.
11. F. Mertens. Über eine zahlentheoretische Function. *Sitzungsber. Akad. Wiss. Wien*, 106(IIa):761–830, 1897.
12. G. Neubauer. Eine empirische Untersuchung zur Mertensschen Funktion. *Numer. Math.*, 5:1–13, 1963.

13. A.M. Odlyzko and H.J.J. te Riele. Disproof of the Mertens conjecture. *J. Reine Angew. Math.*, 357:138–160, 1985.
14. C. Pomerance, J.W. Smith, and R. Tuler. A pipeline architecture for factoring large integers with the quadratic sieve algorithm. *SIAM J. Comput.*, 17(2):387–403, April 1988.
15. P. Pritchard. Linear prime-number sieves: A family tree. *Sci. Comput. Programming*, 9:17–35, 1987.
16. H.J.J. te Riele. On the history of the function $M(x)/\sqrt{x}$ since Stieltjes. In G. van Dijk, editor, *Thomas Jan Stieltjes - Collected Papers*, volume 1, pages 69–79. Springer-Verlag, 1993.
17. R.D. von Sterneck. Neue empirische Daten über die zahlentheoretische Funktion $\sigma(n)$. In E.W. Hobson and A.E.H. Love, editors, *Proc. of the fifth International Congress of Mathematicians (Cambridge, 22–28 August 1912)*, volume 1, pages 341–343. Cambridge, 1913.
18. R. Tijdeman. Private communication, April 2, 1993.