Proc. Seventh IEEE Int. Symposium on High Performance Distributed Computing, Chicago, IL, July 1998,
IEEE Computer Society Press, Los Alamitos, CA, 1998, pp. 42-49

Computing Twin Primes and Brun's Constant: A Distributed Approach [*]

Patrick H. Fry   Jeffrey Nesheiwat   Boleslaw K. Szymanski

Department of Computer Science
Rensselaer Polytechnic Institute, Troy, NY USA 12180 - 3590
E-mail: {fryp, neshj, szymansk}@cs.rpi.edu

Abstract

This paper describes an implementation of a large heterogeneous distributed parallel computation that counts the distribution of twin primes and calculates Brun's constant and maximal distances between pairs of twin primes. Two primes are twins if they differ by two. It is not known if there are infinitely many twin primes but it was proven that the sum of their inverses converges to the value defined as Brun's constant [2]. Prior to this work, the number of twins and their contribution to Brun's constant was known for all twins up to 10^{14}. *We have advanced this calculation to* $1.8 \cdot 10^{15}$ *and are planning to continue to* 10^{16}.

The computation is distributed using the farmer-workers paradigm. The farmer divides the numerical range into intervals of 10 billion. Workers are assigned intervals to analyze and return results. If a worker fails, e.g., it does not return results within a given time period, the farmer reassigns its intervals to other workers. The farmer also frequently saves its state. If the farmer dies, a new one can be started up with minimal loss of work.

This paper describes the algorithm for locating twin primes, a modified Sieve of Eratosthenes. It also discusses the method for distributing the work to clients, time and space saving optimizations, and multi-platform support. We conclude by presenting some preliminary results up to $1.8 \cdot 10^{15}$ *future directions for the system.*

1. Introduction

A prime number is a positive integer that is evenly divisible by exactly two positive integers: itself and one. Two subsequent odd numbers that are both primes are called twin primes or twins for short. $(3, 5)$, $(5, 7)$, $(11, 13)$, and $(41, 43)$ are all twins.

[*]This work was partially supported by NSF Grant 9527151 and by IBM Corp. The contents of this entry does not necessarily reflect the position or policy of the U.S. Government or IBM Corp.—no official endorsement should be inferred or implied.

Although it is still not known if the set of twins is infinite, Brun [2] proved that the sum of the reciprocals

$$B = (\frac{1}{3} + \frac{1}{5}) + (\frac{1}{5} + \frac{1}{7}) + (\frac{1}{11} + \frac{1}{13}) + (\frac{1}{17} + \frac{1}{19}) + \cdots$$

is convergent; unlike the divergent sum of the reciprocals of all individual primes. B is known as Brun's Constant [1]. This value has been estimated by summing the inverses of twin primes over ever increasing ranges [1, 6, 12] with the current upper limit of 10^{14}. Our goal is to refine this estimate by enumerating twin primes two orders of magnitude further, 10^{16}. In addition to counting the number of twin primes and computing the sum of their inverses, we also find the maximum distance between twins and the location of these gaps. These data are of interest to number theoreticians [12].

This is a formidable computation that would take over 100 years on a single workstation. Parallelism and careful algorithm optimization are necessary to obtain results in more timely manner. The computation is highly parallel and there is no need for synchronization or communication between processes except for providing initial data and gathering results. Hence, a distributed environment is the most economical means of exploiting parallelism in this case.

There have been many attempts to use idle processor power to useful computation. Majority of such system impose too heavy system requirements to be useful for this problem. For example, Legion [9, 10] requires at least two daemons to run on each client and uses a specialized file system. If a client dies, the centralized configuration database must be updated if the client cannot be restarted. Piranha [4, 3] is built on top of the tuple-space based coordination language *Linda* [7]. Piranha implements master-worker parallelism but assumes that the master process is persistent and requires Linda compiler and system on each participating machine. *Virtual BSP Computer* library [11] supports BSP computations over a network of non-dedicated workstations. The system migrates the processes from workstations which became unavailable and supports scalability and tightly synchronized parallel computations. However, it requires an extended BSP library on each participating workstation and cannot survive a crash on the master process.

In contrast, our design goals were (i) to use as many processors as possible, and (ii) to enable a computation to proceed for many months. We also use a farmer-worker paradigm because its most general implementation makes no assumptions about the number of clients or the reliability of connection between client and farmer. The focus of our design was:

Reliability: we allow workers to drop off and join the computation at will, the master can fail, even a checkpoint files can be lost without stopping a progress of the computation.

Portability: we minimize memory, computation and system requirements of a worker and restrict the needed tools to basic standard (e.g., using TCP/IP for communication, gcc for the compiler and standardizing endian and data format for Unix and Linux, ensuring source code portability to NT and Windows95). Minimal host requirements open the problem up to a greater number of potential contributors.

Similar goals and means were used by the DESCHALL project which cracked the code in the RSA DES Challenge[15].

The paper is organized as follows. Section 2 provides details about the twin prime numerical information being collected by the program. Section 3 provides a brief overview of the method of distribution. Sections 4, 5, and 6 describe the components of the distributed system. Section 7 outlines the algorithms used for collecting our results efficiently and accurately. We conclude by presenting some preliminary results up to $1.8 \cdot 10^{15}$.

2. Twin Primes and Brun's Constant

The program collects information about twin primes. Instead of storing each twin, we enumerate them. The 10^{13} twins already found would require approximately 8 bytes of storage or 80000GB total if stored individually. Even with a more efficient storage method that would use one byte per pair (e.g. storing the difference between the twins), 10000GB would still be required.

Brun [2] proved the sum of inverses of all twins is convergent, in contrast with the divergent sum of inverses of individual primes. Prior to this work, the most accurate estimate of this constant was done by Nicely who reported the value $1.9021605778 \pm 2.1 \cdot 10^{-9}$ [12]. Increasing the computation to 10^{16} will increase the accuracy to $\pm 6.24 \cdot 10^{-10}$[1].

In addition to enumerating the twins and calculating the sum of their reciprocals, the maximum distance between twins is stored along with its location in each range of numbers analyzed.

3. Distribution of Application

A client takes an average of 60 minutes to process a range of 10 billion numbers. At that rate, it would take over 100 years for one workstation to reach 10^{16}. Fortunately, this problem is easily parallelizable by dividing the range of numbers to be analyzed among processes. The application requires a distributed environment which supports multiple platforms, has low overhead and is scalable. A server is needed to correlate results. Client - client communication is not required.

A central server, or farmer, is responsible for assigning work to its workers and collecting results. A worker requests work from the farmer, processes the work, and returns the results. Our farmer divides the problem up into *intervals*. One interval is defined as a range of 10 billion. One million of these intervals need to be processed to reach our goal. The farmer stores the results from each interval, along with the worker's IP address, in a file. The farmer and worker are written in C and use TCP sockets for communication. This was chosen both for its reliability and cross-platform support. We use a separate program which works offline to combine the results from all completed intervals.

4. The Farmer

The farmer is responsible for assigning intervals to its workers and collecting results (as described in Section 3). The farmer monitors worker performance to maximize efficiency by dynamically changing the number of intervals assigned to each worker and reassigning intervals if a worker fails to return its results within a specified time period.

4.1. Identifying Workers

The farmer uses an ordered pair, (x, y), to identify each worker. x is the IP address of the worker's host. y is a unique identification number assigned by the farmer. This enables multiple workers to run on the same host (e.g. multiprocessor architectures).

This unique identifier is also useful in failure situations. For example, if the farmer fails and a new one is started, the farmer can differentiate between workers which were processing for the previous farmer and its own workers.

There are at most one million workers needed to finish our computation. Assuming that for each successfully finishing worker, at most 10 fail, we can safely estimate that at most 10 million worker identifiers will be assigned. Each time the farmer is restarted, we add 10^7 to the initial identification number issued to the workers, thereby ensuring that two existing workers will never be assigned the same number.

4.2. Worker Timeouts

The farmer can lose contact with its workers for many reasons: network latency, failures on the workstation running the worker, or killed workers. The farmer handles all these problems using timeouts. If a worker fails to report in within a fixed time period, the farmer considers it dead and places the assigned intervals back in the ready queue along with all intervals waiting to be processed. If the worker reports in after this timeout period, it is not assigned any more work and is told to shut itself down. If the worker takes too long sending its finished work or receiving new work from the farmer, the worker shuts itself down. In this way we avoid receiving duplicate messages from the workers or assigning duplicate work.

4.3. Checkpoints

For every 200 messages received containing interval results, the farmer saves its state to a checkpoint file. This value of checkpoint frequency was selected as a compromise between the cost of checkpointing and the cost of farmer's failure. Each message contains an average of 3 intervals. Therefore a farmer crash would, on average, cause the loss of only 300 intervals. Each interval which has not yet been processed, either because it has not been assigned to a worker or is still currently being processed, is stored in the checkpoint file. Intervals which have already been processed are stored in another file which contains all completed work. If the farmer fails, it can be restarted in *recovery mode* by reloading its state from the checkpoint file. This enables processing to continue with only a few hours of lost work. Two checkpoint files are kept at all times. If the farmer crashes during the checkpoint process, the prior checkpoint file may be used to restart the farmer. The checkpoint files and processed intervals file are archived daily.

5. Workers

The worker is where the actual twin prime computations take place. Upon startup, the worker connects to the server where it receives a unique identification number, and a limit specifying how far to enumerate twin primes. Using this limit, the worker reads $\sqrt{limit}$ primes from a binary file that contains all primes up to 10^8. The worker is then assigned several intervals of 10 billion over which it must compute the number of twins, Brun's constant, and the maximum distance between twins.

The number of intervals assigned is dynamically set based on the response time of the worker. A window of 2.5 - 3.5 hours is defined as *acceptable*. Should the worker take more time to return its results,, the number of assigned intervals is halved. Likewise, should the worker respond too soon, the number intervals is increased by one . This ensures that the worker will communicate with the server roughly once every 3 hours.

5.1. Heterogeneity

Many problems arise when dealing with a mathematically intensive heterogeneous distributed application. Most portability issues revolve around the fact that the program relies on 64 bit integers, or `long long` integer types in C. The ANSI standard does not presently support this datatype. As a result, many standard library functions had to be extended to support these longer integers while taking into account endian differences across architectures. For example, functions such as `ntohl()` and `htonl()` which convert long integers between network and host long had to be supplemented with `ntohll()` and `htonll()` routines which do the same for `long long` integers.

Other portability issues became apparent when moving to IRIX, AIX, and Win95/NT platforms. These had to do with signed versus unsigned `char`s being implicit. Furthermore, file I/O issues arose when reading the primes file with the Win32 client. Difficulties with the optimizer on AIX's native C compiler prompted the use of *gcc* for compiling the clients on all platforms. Currently, clients are available on a variety of platforms (Solaris, Sun OS, Linux, FreeBSD, IRIX, HP-UX, Win32, AIX).

5.2. Automated Startup of Workers

It is necessary to find as many hosts to run the worker as possible. Many of the challenges here are not matters of numerical or distributed computing, just system-administration issues which arise when trying to use computers which would otherwise be idle. In some cases, the owner of the machine is also the system administrator, it is easy for them to decide to start up twinprimes as a background process. In other cases, one wants to use machines which are being used for other purposes, and run workers only when completely idle.

At Rensselaer workers are run on a large number of public UNIX workstations, by taking advantage of a separate project under development, called *SCATTERS* [5]. These public workstations are intended for students to use from console. It is expected they will have the full resources of that machine. While these machines were not purchased to solve large-scale numerical problems, they do have the potential to be a large source of cycles when they are idle.

The SCATTERS project is a collection of simple scripts which allow a systems-administrator to start up an arbitrary program (such as the twinprimes worker) on a large number of workstations. The scripts are currently specific to Rensselaer's computing environment, but the ideas are simple enough to adapt to other environments.

There is one script which finds out which hosts are currently idle by parsing the output of standard UNIX commands such as `finger` and `rusers`. There is second script which takes a list of hosts and a single UNIX command as parameters. It uses a slightly modified version of `ssh` (as opposed to `telnet`) to log into the specified hosts and execute the given command. A third script iteratively calls the second script. All of these scripts run on a single workstation, which is the *control station* for all of the SCATTERS processing.

Automated execution is started by running a perl script which does two things. It starts a program (a worker, in this case), and every 10 seconds it checks to see if a user has logged into the workstation. Upon log in, the script will terminate the program and log off the system. Within seconds, all of the resources of the workstation are available to the console user.

A system administrator sits down at the *control station* and types one command:

```
  scatters_loop -count 200 -sleep 1800
-pubibm -pubsgi -pubsun
```

Workers are automatically started on as many as 245 workstations for the next 100 hours, when those machines are idle. The real limit to this `scatters_loop` script is how long the *control station* stays up before it needs to be rebooted (and that has generally been for weeks at a time). This is one of the largest single sources of cycles for the twin primes project.

6. The Relay

The relay enables workers to communicate with the farmer through a firewall. This relay receives worker messages, forwards them to the farmer, and sends the farmer's response back to the worker. This has proven useful for *Beowulf* class machines [14] in which only one node is accessible outside the cluster. The relay runs on the gateway node which has access outside the cluster. In a network with a firewall, the relay resides on the firewall.

We have also been investigating using WWW proxies to forward worker messages to the farmer. Enabling a WWW proxy to forward worker traffic to our farmer would eliminate the need for the relay.

7. The Algorithm

A memory efficient method is required for storage of the 6 million primes used for the sieve and the range of numbers being analyzed. Section 7.1 describes these data storage techniques. Section 7.2 details the algorithm used to keep the calculation of Brun's constant accurate.

7.1. Data Representation

The program uses the Sieve of Eratosthenes method to locate twin primes. All primes up to the square root of the limit must be known. For 10^{16}, all primes up to 10^8 are stored.

As the program has evolved, so has the method of storing these primes. Initially, the actual primes were stored in an array. There are just under 6 million primes, so the program used 24MB to store them. Such a memory requirement would have been a serious restriction on the number of machines which could participate in the computation. This memory requirement was reduced by using just one byte per prime by storing the difference between the current prime and its previous prime; thus using only 6MB. This is the implementation used by Nicely [12].

Currently, one bit for each *potential* prime up to 10^8 is used. Eliminating multiples of 2, 3, and 5 leaves only 8 potential primes for each 30 numbers. As shown in Figure 1, the potential primes are 1, 7, 11, 13, 17, 19, 23, and 29, modulo 30. Memory usage is reduced to $10^8/30 \approx 3.3$MB, halving the memory requirement from storing the distances.

There are 8 potential primes modulo 30, but only three pairs of these can be twins (circled in Figure 1). It is impossible for any number which is 7 or 23 modulo 30 to be a twin because the numbers distance 2 below and above are always multiples of 3 and 5. In prior implementations each odd number was treated as a potential twin member, 15 bits per 30 numbers. Now only 3 bits are used for every 30 numbers. And all multiples of 2, 3 and 5 are automatically removed.

With all multiples of 2, 3 and 5 eliminated, the program then eliminates all the multiples for all the remaining primes up to 10^8 (this process is referred to as shooting out). Smaller primes have more multiples to shoot out. We use a method which shoots out the multiples of smaller primes for a small range of numbers and then *preshoots* these primes out of future ranges. This is done by dividing the range of numbers into *shooting galleries* whose size is divisible by all smaller primes. Currently we use galleries of size

$$30 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 = 223092870$$

which occupies about 2.8MB of memory. This is the largest gallery which can be indexed by 4 byte integers. This allows us to preshoot multiples of 7, 11, 13, 17, 19 and 23 which are preshot in a bit pattern that is kept in memory. Every new gallery is created by bit copying from the bit pattern.

Consider $7^2 = 49$. This number would be shot out in the bit pattern (eliminating a potential 17, 19 twin pair). Table 1 shows some multiples which will automatically be shot out of future galleries.

Preshooting these six primes, in addition to removing multiples of the first three primes (2,3,5), saves processing time. While only the first nine primes are eliminated out of

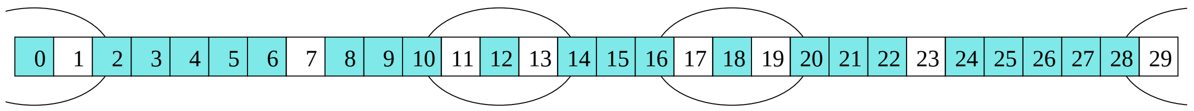

Figure 1. Potential primes and twins modulo 30. Multiples of 2, 3 and 5 are grayed out and potential twins are circled.

$$\begin{aligned} 223092870 + 49 &= 223092919 \\ (2 \cdot 223092870) + 49 &= 446185789 \\ (3 \cdot 223092870) + 49 &= 669278659 \\ &\vdots \end{aligned}$$

Table 1. Multiples of seven are preshot out of the galleries.

the millions of primes which have to be examined, these nine are the ones which would have required the most processing had they been manually shot out of each gallery. Preshooting more primes would require eight byte integers for indexing in the galleries.

To estimate the savings attributable to preshooting, it is easy to notice that the work that a prime causes is proportional to its inverse which is proportional to the number of prime's multiples in a large range. Hence, the work of preshot primes is $\sum_{i=4}^{9} \frac{1}{p_i} = 0.4652\ldots$, where p_i denotes i-th prime (counting 2 as the first prime). We verified that $\sum_{i=10}^{100,000} \frac{1}{p_i} = 1.41\ldots$, yielding an approximation for $\sum_{i=10}^{5761455} \frac{1}{p_i} \approx 1.64$ ($p_{5761455} = 99,999,989$ is the last prime $\leq 10^8$, so the last prime that our algorithm will process). Hence, without preshooting, the computation would take about 30% longer. Estimating from the source code and the time it takes to compute a single range of 10 billion numbers, the program spends about 10 instructions or 20 cycles to shoot out a multiple, so the total computation will require 300 Tera cycles. We processed so far about 60 Tera cycles which otherwise will be idle (we are using floating point operations sparingly so, we cannot measure our progress in Teraflops).

7.2. Accuracy in Computation of Brun's Constant

The main difficulty in this part of the computation is to maintain the numerical precision of the result. Potentially, each floating point operation contributes an error to the result, so after each inverse operation, we know the result with the relative error $\pm 2^{-55}$ and each addition adds an error of the same magnitude. We want to add inverses of primes up to 10^{16} and there are about 10^{12} of them, so the error would potentially add up to leave only four digits of precision out of the initial 16 decimal digits of precision. Of course, with high probability, the error will be smaller (because some errors will cancel each other out). Still this is not a satisfactory solution.

Fortunately, we can use already known values of Brun's constant for the initial primes up to $b = 10^{14}$. Hence, we need to compute only an addition to Brun's constant representing the sum of inverses of twins bigger than b. This addition is of magnitude of at most 10^{-3}, thus adding three digits to the number of significant digits of the result (Brun's constant is about 2).

To further increase precision, we can use long division operations to produce a 32 digit result. The procedure for such long division uses two floating point divisions to create 16 digits of the result each and four long integer multiplications to compute a precise remainder from the first division. Hence, it is rather costly operation. However, the resulting precision of the final result (Brun's constant addition) is 34 digits if the multiword addition is used.

To avoid direct summation, to increase the number of significant digits and to decrease the number of long divisions, we can use approximations in which some of the summations are replaced by a single operation of multiplication and addition of some small corrective value. Only few digits of the corrective value will impact the result, so loss of precision in the correction is irrelevant.

Let $[s, s+r]$ where $r < s/2$ denote the range in which we compute a partial sum of Brun's constant, n denotes the number of twins in this range and $s+t_1, s+t_2, \ldots, s+t_n$ are the subsequent twins. Of course $0 \leq t_1 \leq t_2 \ldots \leq r$. Then, for some $0 < c < r$ the partial sum which we want to compute is

$$P(s,r) = \sum_{i=1}^{n} \frac{1}{s+t_i} = \frac{1}{s+c} \sum_{i=1}^{n} \frac{1}{1 + (t_i - c)/(s+c)}$$

However,

$$\frac{1}{1+x} = 1 - x + x^2 \ldots + (-x)^k \ldots$$

and

$$|\frac{1}{1+x} - \sum_{i=0}^{k}(-x)^k| = |\frac{x^{k+1}}{1-x}| < 2|x^{k+1}|$$

for $-1/2 < x < 1/2$ which is satisfied for $x = (t_i - c)/(s + c)$ because $c < r$ implies $x > -r/(s+r) > -r/s > -1/2$ and $c > 0$ yields $x < r/s < 1/2$.

We will keep partial results as integers (the floating point results of some operations will be multiplied by the proper power of 10 and converted into a long long integer). This representation adds digits of precision to the result if the computed partial sums are added using multi-word arithmetic. The exact number of additional digits will depend on the absolute value of the result, which on the other hand is a function of b.

$$P(s,r,c) \approx \frac{1}{s+c}\left(n - \frac{1}{s+c}\left(\sum_{i=0}^{n}(t_i - c) - \frac{\sum_{i=1}^{n}(t_i-c)^2}{s+c}\right)\right)$$

The first term's numerator (i.e. n) can be computed exactly and using long division described above we can get 32 digits of precision of the result.

The second term will be minimized if we select an integer $c = \sum_{i=1}^{n} t_i/n$. The sum $\sum_{i=1}^{n} t_i/n$ can be computed exactly for $r < 10^9$ in long long integer. With this choice of c, the second term is less than n/s^2, so it can be represented as an integer. The third term is smaller than $n \cdot r^2/s^3$ and for $r < 10^8$ the summation can be computed exactly as an integer.

Finally, we ignore all the higher terms, which contributes to the numerical error of the result. The magnitude of this error can be estimated as at most twice the sum of fourth terms which in turn is less than $2 \cdot n \cdot r^3/s^4$.

Using Nicely's result[12], we can set $b = 10^{14}$ and the final precision of the sum of inverses will be at least 30 decimal digits for $r = 3003$ (for convenience, we select the size that is the divisor of the gallery size). Using the alternative method of performing long division for each pair of twins, we could get 34 decimal digits of precision, but with 3000 times more long divisions executed which would than dominate the computation time of the whole algorithm.

8. Results

The largest range for which all twin primes were found has been growing together with capabilities of computers [1, 6, 12] and currently is 10^{14} [12]. As of the writing of this paper, spare cycles on over 325 computers have been used to compute twin primes up to $1.8 \cdot 10^{15}$, Thus far, 2046397127045 twins have been found. Using these values, Brun's Constant has been computed to be 1.826594298366498. Additionally, the maximum distance between two twins is 25230 and occurs at the twin starting at (1149418981410179, 1149418981410181). These results were obtained over the course of 4 months and continue to be updated.

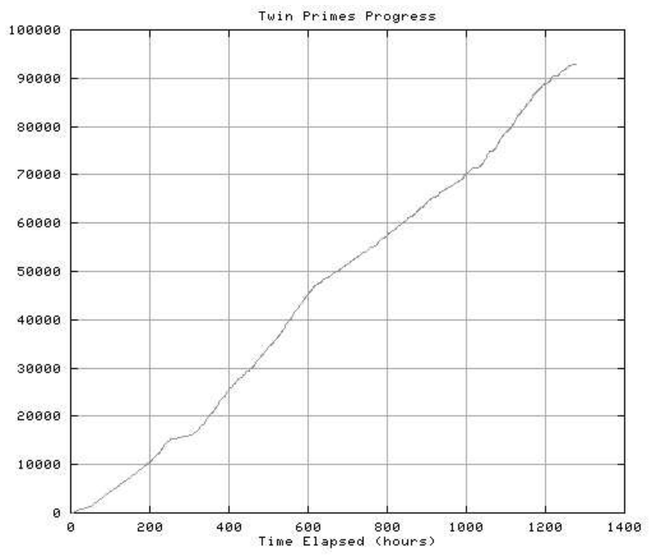

Figure 2. Progress of twin primes computation from 0 to 1e15. (time vs number of intervals).

Figure 2 shows the progress made from start to 10^{15}. The slope flattens out between 200 and 400 hours into the computation. This is the result of a server crash which had to be restarted in *recovery mode*. The slow ramp up is the result of clients being started up manually. The slope decreases again 600 hours into the computation when 16 off-site *Beowulf* nodes were suspended for benchmarking purposes.

Figure 3 shows the progress from 10^{15} to $1.8 \cdot 10^{15}$. During this time, the server has not crashed at all. At approximately 100 hours into the computation, one of the relay agents went down and is evident by the change in slope. As the computation progressed, the slope slowly decreased followed by a sharp increase after 1100 hours. This trend is due to the academic calendar at Rensselaer Polytechnic Institute. The majority of clients run on public workstations when they are idle. As the semester progressed, the workstations were in use most of the time and rarely idle. The sharp increase in slope coincides with the end of classes hence the idle workstations were tasked with computing twin primes.

9. Conclusions

While the current record has been surpassed, our goal is to compute all twins up to 10^{16}. At the current rate with approximately 325 clients, this goal should be reached in a

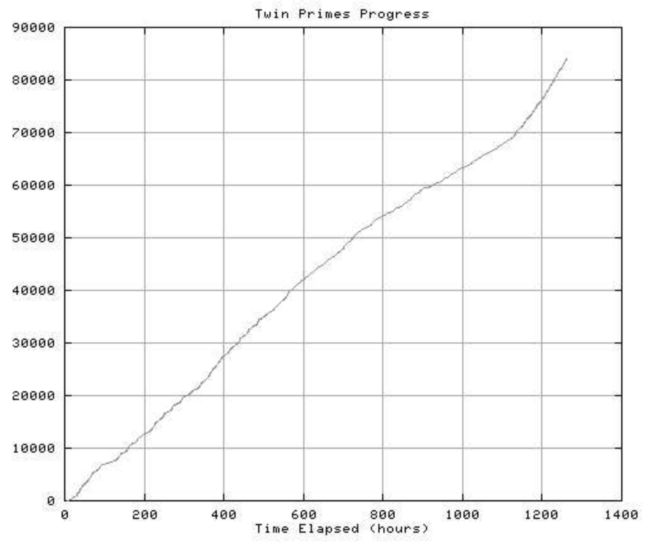

Figure 3. Progress of twin primes computation from 1e15 to 1.8e15. (time vs number of intervals).

year.

9.1 Future Work

Extensions to this work involve furthering the existing twin primes computation and also applying this fault tolerant distributed computing framework to other similar problems. In order to maximize utilization of idle workstations, our attention has been focused on the Windows 95/NT client. This client is unique in that it cannot be started and stopped remotely in the same way as the UNIX clients. We are looking into two solutions to this problem. The first involves initiating the worker when the screensaver engages and suspending when the screensaver disengages. The other solution will require developing a specific service that listens on a port for worker start and stop requests that come in remotely, as is currently the case for clients running on other platforms. The advantege of the latter solution is that it can be applied to a large class of mathematical, enumeration, and search problems.

The mechanism we have employed for enumerating twin primes is generic in that the farmer assigns portions of a problem to many workers while maintaining a high degree of fault tolerance and ensuring recoverability. The problem specific computations are done by the worker. Future plans include making the farmer and its accompanying communication protocol generic so workers can be developed to solve a range of computationally intensive problems in number theory, encryption and other areas. To widen the range of the applications to problems whose computation require synchronization, we envision the following extension of our protocol.

In the first stage of the synchronous step n workers will initiate computation and by a time-out, $p \leq 1.0$ fractions of them will finish. The remaining uncomputed $n(1-p)$ chunks are then computed by survivors with about $\frac{1}{1-p} - 1$ workers assigned to each chunk. As a result, with high probability $1-(1-p)^{\frac{1}{1-p}-1}$ the step will finish or we will initiate the third stage with even higher replication level (and so higher probability of finishing). For example, in our computation $n = 200$, $p = 1/5$, so in 99.84% cases we will be able to finish in two stages, yielding a respectable efficiency of 50% compared to non-synchronous case. A maximum independent set problem is an example of a synchronous parallel computation [8] with a low communication requirement if proper data distribution is used [11]. We plan to use our system with a protocol extended for synchronous computation to this problem.

9.2 Call for Cycles

With your help, we can reach our goal sooner. Clients are available for the following platforms:

- AIX4
- HP-UX
- Linux (elf) and FreeBSD
- IRIX and Cray Origin 2000
- SunOs 4.1.3 and Solaris 2.5 and 2.6
- Windows 95/NT

All of the information needed to contribute spare CPU cycles to this effort is available at *http://www.cs.rpi.edu/research/twinp*

To date, the greatest contributors are ranked in Table 2. The `rpi.edu` machines include the public computing labs at Rensselaer. `cs.rpi.edu` include the Computer Science departmental workstations and `momentum.cs.rpi.edu` is a 12 processor Cray Origin 2000. The remaining entries represent students' personal computers, their contribution was significant during the school year and has dropped off as the semester drew to a close. Each *CHUNK* represents an interval of 10 billion.

9.3. Acknowledgments

The authors wish to thank the following people for their support and invaluable assistance: Garance A. Drosehn

POS	CHUNKS	DOMAIN / GROUP
1	35992	rpi.edu (247 hosts)
2	22458	cs.rpi.edu (40 hosts)
3	19674	momentum.cs.rpi.edu
4	3493	nycap.rr.com (5 hosts)
5	1823	stu.rpi.edu (3 hosts)

Table 2. Ranking of top 5 contributors.

(RPI-CIS) for automating startup of clients on public workstations using SCATTERS [5]. Nathan Schimke (RPI-CS) for automating startup on CS department workstations. Tom Cwik (JPL) for access to HPCC resources, Josh Wilmes (RPI-CS) for assistance in porting to AIX, Robert Dugan (RPI-CS) for assistance with porting to win95/NT.

Thank go to everyone who has supported our efforts by donating spare CPU cycles or by providing useful feedback, in particular: David Armstrong (RPI), James Bonanno (RPI), Will Coleda (UT), Erik Collin (EIS), Barnaby Court (RPI), Steve Czetty (RPI-JPL), Dan Daglas (Citicorp), Brett Hogden (RGE), Thomas Jennings (RPI), Simon Karpen (RPI), Tim Klink (UT), Mat Maessen (UT), Matthew Schnee (Digital), Chris Stevens (RPI), Kim Stevens (RPI-CS), Dave Thompson (RPI).

References

[1] R. P. Brent. Irregularities in the distribution of primes and twin primes. *Math. Comp.*, 29(129):43–56, 1975.

[2] V. Brun. La série 1/5 + 1/7 + 1/11 + 1/13 + 1/17 + 1/19 + 1/29 + 1/31 + 1/41 + 1/43 + 1/59 + 1/61 + ... où les dénominateurs sont 'nombres premieres jumeaux' est convergente ou finie. *Bull. Sci. Math.*, 43:124–128, 1919.

[3] N. Carriero, E. Freeman, D. Gelernter, and D. Kaminsky. Adaptive parallelism and piranha. *Computer*, 28(1):40–49, 1995.

[4] N. Carriero, D. Gelernter, D. Kaminsky, and J. Westbrook. Adaptive parallelism with Piranha. Technical Report 954, Yale University, 1993.

[5] G. A. Drosehn. SCATTERS - a Simple Cool Admin Tool To Everywhere Run Something. http://www.rpi.edu/∼drosehn/Projects/SCATTERS.html.

[6] C. E. Fröberg. On the sum of inverses of primes and twin primes. *Nordisk Tidskr. Informationsbehandling (BIT)*, 1:15–20, 1961.

[7] D. Gelernter, M. Jourdenais, and D. Kaminsky. Piranha scheduling: Strategies and their implementation. Technical Report 983, Yale University, 1993.

[8] M. K. Goldberg and D. L. Hollinger. Database learning: a method for empirical algorithm design. In *Proc. Workshop on Algorithm Engineering*, 1997.

[9] A. S. Grimshaw, A. Nguyen-Tuong, and W. A. Wulf. Campus-wide computing: Early results using legion at the University of Virginia. Technical report, University of Virginia, 1995.

[10] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. R. Jr. A synopsis of the Legion project. Technical report, University of Virginia, 1994.

[11] M. Nibhanupudi and B. K. Szymanski. Runtime support for virtual bsp computer. In *Proc. Workshops at 12th Intern. Parallel Processing Symposium*. Springer Verlag, 1998.

[12] T. R. Nicely. Enumeration to 1e14 of the twin primes and Brun's constant. *Virginia Journal of Science*, 46(3):195–204, Mar. 1996.

[13] M. J. Quinn. *Parallel Computing: Theory and Practice*. McGraw-Hill, Inc., 1994.

[14] T. Sterling, D. Becker, J. Salmon, D. Katz, P. Angelino, D. Ridge, and J. Lindheim. How to build a Beowulf: A tutorial, Oct. 1997. Presented by the Center for Advanced Computing Research, California Institute of Technology.

[15] R. Verser. The $10,000 DES challenge. http://www.frii.com/∼rcv/deschall.htm.

[16] A. Y. Zomaya, editor. *Parallel & Distributed Computing Handbook*, chapter 5. McGraw-Hill, Inc., 1996.