**The Fermat Number Factor Search Program “pmfs”**

**Version 3.0**

**Background**

The "pmfs" program is a Parallel Multi-precision Fermat factor Search program. Fermat numbers have the form Fm = 2^2^m + 1. Factors of Fermat numbers must have the form k \* 2^n + 1, where k is odd and n >= m+2.

pmfs tests each trial divisor k \* 2^n + 1 within a specified range of n and k to determine if it divides any Fm where m <= n-2. For each n in the range, pmfs will search the range of k, then move on to the next value of n. The range of k is divided into subranges called k groups. For each k group, pmfs will use a sieve to eliminate the trial divisors that are divisible by a small primes. Surviving trial divisors will then be tested to determine if Fm mod k \* 2^n + 1 = 0 for each m <= n-2.

To efficiently use multiple processor cores within a system, pmfs has one master thread and multiple slave threads. The master thread reads the input file specifying search ranges, launches the specified number of slave threads, assigns k groups to the slave threads to search, prints results including factors found, and terminates all slave threads when the search range is complete. While all slave threads are busy searching k groups, the master thread periodically polls the slave threads for results, then sleeps in order to make all cores available to the slave threads.

pmfs uses the GMP library and runs on an x86-64 Linux system (GMP\_LIMB\_BITS = 64). It has been tested on Red Hat, SUSE and Ubuntu. Currently there is not a version of pmfs for Windows.

**Installing and running pmfs**

Unzipping the pmfs\_3.0.zip file will produce this word document, the performance benchmark spreadsheet and a tar file. Move the tar file to an x86-64 Linux system and un-tar it. The tar file contains the executable (pmfs\_3.0), a symbolic link (pmfs) and example search ranges (test2.in) and results (test2.out) files. pmfs reads the search ranges file from stdin and writes results to stdout. pmfs currently has no command line flags. A typical run command is:

pmfs < test2.in > test2.out

Here is an example search ranges input file (test2.in):



Lines starting with # and blank lines are ignored. Multiple search ranges can be specified, one range per line. Each range is specified using the following parameters (all must be present). Commas are allowed in the ki, kf and k\_grp fields for readability. Only odd values of k are tested.

|  |  |
| --- | --- |
| **Column** | **Description** |
| ni | The initial n of the search range. |
| nf | The final n of the search range. |
| n\_step | The increment of n. Used primarily for benchmarking. Example: ni = 1000 and n\_step = 100 will search the specified range of k for n = 1000, 1100, 1200, etc. |
| ki | The initial k of the search range. ki must be odd. |
| kf | The final k of the search range. kf must be even. |
| k\_grp | The number of k values in the k group. |
| n\_primes | The number of small primes to use in the sieve. |
| n\_threads | The number of slave threads to launch. Usually set to the number of processor cores in the system. |

The results output file is currently fairly verbose, with a line being printed for each N start, N done and K group done. Unwanted information can be removed from the output file using “fgrep –v” after the run is complete. Factors will be printed as they are found, and can be displayed using:

fgrep divides test2.out

If the system crashes or pmfs is killed, the run can be restarted by modifying the search ranges to start from the next n beyond the last “N done” printed in the results file.

**Performance**

The GMP-Fermat program written by Mark Rodenkirch is currently one of the fastest Fermat factor search programs, as shown in <http://www.fermatsearch.org/productivity.html>. GMP-Fermat has two executables: fermat64 which uses only the GMP library, and fermat64\_redc which also uses optimized assembly routines. The performance of pmfs version 3.0 and GMP-Fermat version 2.5 were tested on an Ivy Bridge E7-8890 v2 system with various values of n and n\_primes. The results are shown in the benchmarks spreadsheet. Here is a quick summary:

* GMP-Fermat is faster than pmfs for n <= 500 by anywhere from 1.1x to 7.1x, while pmfs is faster than GMP-Fermat for n >= 1000 by anywhere from 1.1x to 11.8x. This is when running pmfs with one core, and the range covers both GMP-Fermat executables, the value of n and the number of primes in the sieve.
* GMP-Fermat uses a much faster sieve setup algorithm than pmfs, and can therefore use many more primes effectively (500,000 in my testing). pmfs is limited to about 50,000 primes. However, using the same number of primes, more trial divisors survive the sieve in GMP-Fermat than in pmfs. (I sent Mark a test case to look at.)
* pmfs performance scales nearly linearly with the number of cores (n\_threads) used. Scaling was generally in the 200x to 220x range for 240 cores compared to 1 core.

I believe the single core performance differences between fermat64 and pmfs come primarily from the algorithms used to perform square/mod. fermat64 performs square/mod using mpz\_powm, which calculates r = r^2 mod (k \* 2^n + 1) using a single GMP function call. This is fast for small n <= 500, but I believe the division by the full k \* 2^n + 1 becomes time consuming for large n >= 1000. pmfs squares using mpz\_mul, then calculates mod (k \* 2^n + 1) by dividing (r^2 / 2^n) by k followed by several shift and add/sub operations. While this is slower for small n, it appears to be more efficient for large n.

**What’s next?**

Doing this benchmarking has motivated me to start working on a new version of pmfs. I have a long list of improvements I would like to make. While pmfs is not currently open source, if anyone uses pmfs and has suggestions for changes / additions, please post them to the pmfs thread on mersenneforum.org / Factoring Projects / FermatSearch and I will try to include them.