­­­Title: Raspberry Pi 2B+ runs FFT on GPU and CPU, energy and power comparison

Author: Qihao He, Bruce Segee, Vincent weaver

# GPU part:

Library name: GPU\_FFT

Written by: Andrew Holme

## Library description:

Reference:

ACCELERATING FOURIER TRANSFORMS USING THE GPU

<https://www.raspberrypi.org/blog/accelerating-fourier-transforms-using-the-gpu/>

General purpose code for the VideoCore IV graphics processing unit(GPU) in the BCM2835. To create an accelerated Fast Fourier Transform library. Taking the Fourier transform of a function yields its frequency spectrum (i.e. the pure harmonic functions which can be added together to reconstruct the original function).

GPU\_FFT is an FFT library for the Raspberry Pi which exploits the BCM2835 SoC V3D hardware to deliver ten times the performance that is possible on the 700 MHz ARM. Kernels are provided for all power-of-2 FFT lengths from 256 to 131,072 points inclusive.

GPU\_FFT uses single-precision floating point for data and twiddle factors, so it does not compete on accuracy with double-precision libraries; however, the relative root-mean-square (rms) error for a 2048-point transform is less than one part per million, which is not bad.

The library runs on dedicated 3D hardware in the BCM2835 SoC, and communication between ARM and GPU adds 100µs of latency which is much longer than the shortest transform takes to compute! To overcome this, batches of transforms can be executed with a single call. Typical per-transform runtimes in microseconds are:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Points | batch=1 | batch=10 | batch=50 | FFTW | Speedup |
| 256 | 112 | 22 | 16 | 92 | 5.8x |
| 512 | 125 | 37 | 26 | 217 | 8.3x |
| 1024 | 136 | 54 | 45 | 482 | 10.7x |
| 2048 | 180 | 107 | 93 | 952 | 10.2x |
| 4096 | 298 | 256 | 240 | 3002 | 12.5x |
| 8192 | 689 | 624 | 608 | 5082 | 8.4x |
| 16384 | 1274 | 1167 | 1131 | 12005 | 10.6x |
| 32768 | 3397 | 3225 | 3196 | 31211 | 9.8x |
| 65536 | 6978 | 6703 | 6674 | 82769 | 12.4x |
| 131072 | 16734 | 16110 | 16171 | 183731 | 11.4x |

## ­Library usage:

Being able to perform lots of Fourier transforms quickly is useful for all sorts of audio and radio applications including, unsurprisingly, GPS. Ham radio enthusiasts will also find Andrew’s work very useful.

Last October, Eben attended the Radio Society of Great Britain (RSGB) Convention, where radio amateurs told him they wanted a speedy fast Fourier transform (FFT) library to do Software Defined Radio (SDR) projects on the Pi.

## Library operate:

To get GPU\_FFT enter the following at the command prompt:

sudo rpi-update && sudo reboot

To build and run the example program:

directory:

cd /opt/vc/src/hello\_pi/hello\_fft

make

sudo mknod char\_dev c 100 0

sudo ./hello\_fft.bin

API documentation can be found in the hello\_fft folder.

File: hello\_fft usage:

$ sudo ./hello\_fft.bin

Usage: hello\_fft.bin log2\_N [jobs [loops]]

log2\_N = log2(FFT\_length), log2\_N = 8...22

jobs = transforms per batch, jobs>0, default 1

loops = number of test repeats, loops>0, default 1

e.g.

$ sudo ./hello\_fft.bin 8

rel\_rms\_err = 3.3e-07, usecs = 45, k = 0

$ sudo ./hello\_fft.bin 22

rel\_rms\_err = 1.5e-06, usecs = 782379, k = 0

$ sudo ./hello\_fft.bin 8 2 2

rel\_rms\_err = 3.1e-07, usecs = 33, k = 0

rel\_rms\_err = 3.1e-07, usecs = 23, k = 1

## Library details:

Reference: gpu\_fft.txt from the library hello\_fft

BCM2835 "GPU\_FFT" release 3.0 by Andrew Holme, 2015.

GPU\_FFT is an FFT library for the Raspberry Pi which exploits the BCM2835 SoC

3D hardware to deliver ten times more data throughput than is possible on the

700 MHz ARM of the Pi 1. Kernels are provided for all power-of-2 FFT lengths

between 256 and 4,194,304 points inclusive. A transpose function, which also

uses the 3D hardware, is provided to support 2-dimensional transforms.

\*\*\* Accuracy \*\*\*

GPU\_FFT uses single-precision floats for data and twiddle factors. The output

is not scaled. The relative root-mean-square (rms) error in parts-per-million

(ppm) for different transform lengths (N) is typically:

log2(N) | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15

ppm rms | 0.33 | 0.46 | 0.52 | 0.59 | 0.78 | 0.83 | 0.92 | 0.98

log2(N) | 16 | 17 | 18 | 19 | 20 | 21 | 22

ppm rms | 1.0 | 1.3 | 1.3 | 1.4 | 1.5 | 1.5 | 1.5

Accuracy has improved significantly over previous releases at the expense of a

small (2%) performance hit; however, FFTW is still one order of magnitude more

accurate than GPU\_FFT.

\*\*\* Throughput \*\*\*

GPU\_FFT 1.0 had to be invoked through a "mailbox" which added a 100us overhead

on every call. To mitigate this, batches of transforms could be submitted via

a single call. GPU\_FFT now avoids this 100us overhead by poking GPU registers

directly from the ARM if total batch runtime will be short; but still uses the

mailbox for longer jobs to avoid busy waiting at 100% CPU for too long.

Typical per-transform runtimes for batch sizes of 1 and 10; and comparative

figures for FFTW (FFTW\_MEASURE mode) on a Pi 1 with L2 cache enabled are:

log2(N) | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15

1 | 0.033 | 0.049 | 0.070 | 0.12 | 0.25 | 0.61 | 1.2 | 3.5

10 | 0.017 | 0.029 | 0.049 | 0.11 | 0.27 | 0.66 | 1.2 | 3.3

FFTW | 0.092 | 0.22 | 0.48 | 0.95 | 3.0 | 5.1 | 12 | 31

log2(N) | 16 | 17 | 18 | 19 | 20 | 21 | 22 All times in

1 | 7.0 | 17 | 43 | 97 | 194 | 388 | 786 milliseconds

FFTW | 83 | 180 | 560 | 670 | 1600 | 3400 | 8800 2 sig. figs.

\*\*\* API functions \*\*\*

gpu\_fft\_prepare() Call once to allocate memory and initialise data

structures. Returns 0 for success.

gpu\_fft\_execute() Call one or more times to execute a previously

prepared FFT batch. Returns 0 for success.

gpu\_fft\_release() Call once to release resources after use.

GPU memory is permanently lost if not freed.

\*\*\* Parameters \*\*\*

int mb Mailbox file descriptor obtained by calling mbox\_open()

int log2\_N log2(FFT length) = 8 to 22

int direction FFT direction: GPU\_FFT\_FWD for forward FFT

GPU\_FFT\_REV for inverse FFT

int jobs Number of transforms in batch = 1 or more

GPU\_FFT \*\* Output parameter from prepare: control structure.

GPU\_FFT \* Input parameter to execute and release

\*\*\* Data format \*\*\*

Complex data arrays are stored as alternate real and imaginary parts:

struct GPU\_FFT\_COMPLEX {

float re, im;

};

The GPU\_FFT struct created by gpu\_fft\_prepare() contains pointers to the input

and output arrays:

struct GPU\_FFT {

struct GPU\_FFT\_COMPLEX \*in, \*out;

When executing a batch of transforms, buffer pointers are obtained as follows:

struct GPU\_FFT \*fft = gpu\_fft\_prepare( ... , jobs);

for (int j=0; j<jobs; j++) {

struct GPU\_FFT\_COMPLEX \*in = fft->in + j\*fft->step;

struct GPU\_FFT\_COMPLEX \*out = fft->out + j\*fft->step;

GPU\_FFT.step is greater than FFT length because a guard space is left between

buffers for caching and alignment reasons.

GPU\_FFT performs multiple passes between ping-pong buffers. The final output

lands in the same buffer as input after an even number of passes. Transforms

where log2\_N=12...16 use an odd number of passes and the final result is left

out-of-place. The input data is never preserved.

GPU\_FFT and Open GL will run concurrently on Pi 1 if GPU\_FFT is configured not

to use VC4 L2 cache by zeroing a define in file gpu\_fft\_base.c as follows:

#define GPU\_FFT\_USE\_VC4\_L2\_CACHE 0 // Pi 1 only: cached=1; direct=0

Overall performance will probably be higher if GPU\_FFT and Open GL take turns

at using the 3D hardware. Since eglSwapBuffers() returns immediately without

waiting for rendering, call glFlush() and glFinish() afterwards as follows:

## Library results:

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 1 1

rel\_rms\_err = 3.3e-07, usecs = 44, k = 0

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 2 1

rel\_rms\_err = 3.1e-07, usecs = 30, k = 0

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 4 1

rel\_rms\_err = 2.8e-07, usecs = 22, k = 0

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 8 1

rel\_rms\_err = 2.7e-07, usecs = 19, k = 0

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 16 1

rel\_rms\_err = 2.6e-07, usecs = 28, k = 0

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 32 1

rel\_rms\_err = 2.6e-07, usecs = 17, k = 0

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 64 1

rel\_rms\_err = 2.5e-07, usecs = 16, k = 0

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 16 1

rel\_rms\_err = 2.6e-07, usecs = 17, k = 0

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 128 1

rel\_rms\_err = 0.062, usecs = 17, k = 0

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 256 1

rel\_rms\_err = 0.062, usecs = 16, k = 0

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 512 1

rel\_rms\_err = 0.71, usecs = 15, k = 0

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 1024 1

rel\_rms\_err = 0.94, usecs = 15, k = 0

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 2048 1

rel\_rms\_err = 1, usecs = 15, k = 0

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 4096 1

rel\_rms\_err = 1.1, usecs = 15, k = 0

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 8192 1

rel\_rms\_err = 1.1, usecs = 15, k = 0

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 1 4

rel\_rms\_err = 3.3e-07, usecs = 44, k = 0

rel\_rms\_err = 3.3e-07, usecs = 31, k = 1

rel\_rms\_err = 3.3e-07, usecs = 31, k = 2

rel\_rms\_err = 3.3e-07, usecs = 31, k = 3

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 1 8

rel\_rms\_err = 3.3e-07, usecs = 51, k = 0

rel\_rms\_err = 3.3e-07, usecs = 31, k = 1

rel\_rms\_err = 3.3e-07, usecs = 32, k = 2

rel\_rms\_err = 3.3e-07, usecs = 32, k = 3

rel\_rms\_err = 3.3e-07, usecs = 32, k = 4

rel\_rms\_err = 3.3e-07, usecs = 33, k = 5

rel\_rms\_err = 3.3e-07, usecs = 31, k = 6

rel\_rms\_err = 3.3e-07, usecs = 31, k = 7

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 1 16

rel\_rms\_err = 3.3e-07, usecs = 43, k = 0

rel\_rms\_err = 3.3e-07, usecs = 31, k = 1

rel\_rms\_err = 3.3e-07, usecs = 32, k = 2

rel\_rms\_err = 3.3e-07, usecs = 31, k = 3

rel\_rms\_err = 3.3e-07, usecs = 32, k = 4

rel\_rms\_err = 3.3e-07, usecs = 33, k = 5

rel\_rms\_err = 3.3e-07, usecs = 32, k = 6

rel\_rms\_err = 3.3e-07, usecs = 31, k = 7

rel\_rms\_err = 3.3e-07, usecs = 30, k = 8

rel\_rms\_err = 3.3e-07, usecs = 30, k = 9

rel\_rms\_err = 3.3e-07, usecs = 30, k = 10

rel\_rms\_err = 3.3e-07, usecs = 30, k = 11

rel\_rms\_err = 3.3e-07, usecs = 29, k = 12

rel\_rms\_err = 3.3e-07, usecs = 31, k = 13

rel\_rms\_err = 3.3e-07, usecs = 30, k = 14

rel\_rms\_err = 3.3e-07, usecs = 31, k = 15

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 8 1 32

rel\_rms\_err = 3.3e-07, usecs = 46, k = 0

rel\_rms\_err = 3.3e-07, usecs = 31, k = 1

rel\_rms\_err = 3.3e-07, usecs = 32, k = 2

rel\_rms\_err = 3.3e-07, usecs = 31, k = 3

rel\_rms\_err = 3.3e-07, usecs = 32, k = 4

rel\_rms\_err = 3.3e-07, usecs = 31, k = 5

rel\_rms\_err = 3.3e-07, usecs = 32, k = 6

rel\_rms\_err = 3.3e-07, usecs = 30, k = 7

rel\_rms\_err = 3.3e-07, usecs = 31, k = 8

rel\_rms\_err = 3.3e-07, usecs = 29, k = 9

rel\_rms\_err = 3.3e-07, usecs = 29, k = 10

rel\_rms\_err = 3.3e-07, usecs = 29, k = 11

rel\_rms\_err = 3.3e-07, usecs = 29, k = 12

rel\_rms\_err = 3.3e-07, usecs = 30, k = 13

rel\_rms\_err = 3.3e-07, usecs = 32, k = 14

rel\_rms\_err = 3.3e-07, usecs = 31, k = 15

rel\_rms\_err = 3.3e-07, usecs = 30, k = 16

rel\_rms\_err = 3.3e-07, usecs = 30, k = 17

rel\_rms\_err = 3.3e-07, usecs = 30, k = 18

rel\_rms\_err = 3.3e-07, usecs = 29, k = 19

rel\_rms\_err = 3.3e-07, usecs = 30, k = 20

rel\_rms\_err = 3.3e-07, usecs = 30, k = 21

rel\_rms\_err = 3.3e-07, usecs = 29, k = 22

rel\_rms\_err = 3.3e-07, usecs = 29, k = 23

rel\_rms\_err = 3.3e-07, usecs = 29, k = 24

rel\_rms\_err = 3.3e-07, usecs = 29, k = 25

rel\_rms\_err = 3.3e-07, usecs = 30, k = 26

rel\_rms\_err = 3.3e-07, usecs = 29, k = 27

rel\_rms\_err = 3.3e-07, usecs = 29, k = 28

rel\_rms\_err = 3.3e-07, usecs = 29, k = 29

rel\_rms\_err = 3.3e-07, usecs = 29, k = 30

rel\_rms\_err = 3.3e-07, usecs = 31, k = 31

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 22 1 1

rel\_rms\_err = 1.5e-06, usecs = 781473, k = 0

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 22 2 1

rel\_rms\_err = 1.5e-06, usecs = 783190, k = 0

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 22 4 1

Out of memory. Try a smaller batch or increase GPU memory.

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 22 8 1

Out of memory. Try a smaller batch or increase GPU memory.

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 22 2 2

rel\_rms\_err = 1.5e-06, usecs = 785348, k = 0

rel\_rms\_err = 1.5e-06, usecs = 781543, k = 1

pi@raspberrypi:/opt/vc/src/hello\_pi/hello\_fft $ sudo ./hello\_fft.bin 22 2 4

rel\_rms\_err = 1.5e-06, usecs = 781577, k = 0

rel\_rms\_err = 1.5e-06, usecs = 778754, k = 1

rel\_rms\_err = 1.5e-06, usecs = 778890, k = 2

rel\_rms\_err = 1.5e-06, usecs = 778164, k = 3

$ cd /opt/vc/src/hello\_pi/hello\_fft

$ sudo perf\_3.16 stat -d ./hello\_fft.bin 21 4

rel\_rms\_err = 1.5e-06, usecs = 384048, k = 0

Performance counter stats for './hello\_fft.bin 21 4':

16611.682000 task-clock (msec) # 0.884 CPUs utilized

331 context-switches # 0.020 K/sec

0 cpu-migrations # 0.000 K/sec

16,477 page-faults # 0.992 K/sec

11,577,399,763 cycles # 0.697 GHz [37.56%]

1,285,911,727 stalled-cycles-frontend # 11.11% frontend cycles idle [37.54%]

11,624,766 stalled-cycles-backend # 0.10% backend cycles idle [37.36%]

3,590,951,918 instructions # 0.31 insns per cycle

# 0.36 stalled cycles per insn [24.99%]

491,596,173 branches # 29.593 M/sec [25.14%]

12,319,210 branch-misses # 2.51% of all branches [25.08%]

27,138,104 L1-dcache-loads # 1.634 M/sec [25.06%]

697,059 L1-dcache-load-misses # 2.57% of all L1-dcache hits [25.08%]

<not supported> LLC-loads

<not supported> LLC-load-misses

18.790153624 seconds time elapsed

$ cd ~/QH\_directory/userland/host\_applications/linux/apps/hello\_pi/hello\_fft

$ sudo perf\_3.16 stat -d ./hello\_fft.bin 20 10

rel\_rms\_err = 1.4e-06, usecs = 192471, k = 0

Performance counter stats for './hello\_fft.bin 20 10':

20842.337000 task-clock (msec) # 0.812 CPUs utilized

599 context-switches # 0.029 K/sec

0 cpu-migrations # 0.000 K/sec

20,571 page-faults # 0.987 K/sec

14,530,940,586 cycles # 0.697 GHz [37.67%]

1,611,996,079 stalled-cycles-frontend # 11.09% frontend cycles idle [37.42%]

14,560,086 stalled-cycles-backend # 0.10% backend cycles idle [37.28%]

4,522,659,463 instructions # 0.31 insns per cycle

# 0.36 stalled cycles per insn [24.94%]

615,849,834 branches # 29.548 M/sec [25.04%]

14,833,907 branch-misses # 2.41% of all branches [25.11%]

32,706,289 L1-dcache-loads # 1.569 M/sec [25.19%]

1,152,936 L1-dcache-load-misses # 3.53% of all L1-dcache hits [25.16%]

<not supported> LLC-loads

<not supported> LLC-load-misses

25.663854775 seconds time elapsed

$ sudo perf\_3.16 stat -d ./hello\_fft.bin 20 1

rel\_rms\_err = 1.5e-06, usecs = 191712, k = 0

Performance counter stats for './hello\_fft.bin 20 1':

2073.859000 task-clock (msec) # 0.895 CPUs utilized

45 context-switches # 0.022 K/sec

0 cpu-migrations # 0.000 K/sec

2,138 page-faults # 0.001 M/sec

1,445,726,218 cycles # 0.697 GHz [37.45%]

168,054,051 stalled-cycles-frontend # 11.62% frontend cycles idle [37.85%]

2,340,102 stalled-cycles-backend # 0.16% backend cycles idle [38.15%]

439,204,097 instructions # 0.30 insns per cycle

# 0.38 stalled cycles per insn [25.45%]

60,354,270 branches # 29.102 M/sec [25.31%]

1,630,240 branch-misses # 2.70% of all branches [25.16%]

4,900,540 L1-dcache-loads # 2.363 M/sec [25.60%]

155,059 L1-dcache-load-misses # 3.16% of all L1-dcache hits [25.15%]

<not supported> LLC-loads

<not supported> LLC-load-misses

2.316615205 seconds time elapsed

$ sudo perf\_3.16 stat -d ./hello\_fft.bin 20 10 1

rel\_rms\_err = 1.4e-06, usecs = 192392, k = 0

Performance counter stats for './hello\_fft.bin 20 10 1':

20795.216000 task-clock (msec) # 0.820 CPUs utilized

622 context-switches # 0.030 K/sec

0 cpu-migrations # 0.000 K/sec

20,571 page-faults # 0.989 K/sec

14,462,479,493 cycles # 0.695 GHz [37.65%]

1,591,452,238 stalled-cycles-frontend # 11.00% frontend cycles idle [37.42%]

14,030,599 stalled-cycles-backend # 0.10% backend cycles idle [37.63%]

4,527,389,098 instructions # 0.31 insns per cycle

# 0.35 stalled cycles per insn [25.28%]

618,832,550 branches # 29.758 M/sec [25.06%]

14,727,795 branch-misses # 2.38% of all branches [24.83%]

36,632,522 L1-dcache-loads # 1.762 M/sec [24.99%]

997,478 L1-dcache-load-misses # 2.72% of all L1-dcache hits [25.00%]

<not supported> LLC-loads

<not supported> LLC-load-misses

25.369885801 seconds time elapsed

$ sudo perf\_3.16 stat -d ./hello\_fft.bin 20 10 2

rel\_rms\_err = 1.4e-06, usecs = 193745, k = 0

rel\_rms\_err = 1.4e-06, usecs = 194533, k = 1

Performance counter stats for './hello\_fft.bin 20 10 2':

41489.336000 task-clock (msec) # 0.813 CPUs utilized

1,241 context-switches # 0.030 K/sec

0 cpu-migrations # 0.000 K/sec

20,567 page-faults # 0.496 K/sec

28,924,373,678 cycles # 0.697 GHz [37.64%]

3,146,747,530 stalled-cycles-frontend # 10.88% frontend cycles idle [37.61%]

27,314,821 stalled-cycles-backend # 0.09% backend cycles idle [37.43%]

9,032,409,981 instructions # 0.31 insns per cycle

# 0.35 stalled cycles per insn [24.85%]

1,234,499,221 branches # 29.755 M/sec [24.84%]

29,373,667 branch-misses # 2.38% of all branches [25.05%]

59,944,125 L1-dcache-loads # 1.445 M/sec [25.11%]

2,015,937 L1-dcache-load-misses # 3.36% of all L1-dcache hits [25.19%]

<not supported> LLC-loads

<not supported> LLC-load-misses

51.033239728 seconds time elapsed

$ sudo perf\_3.16 stat -d ./hello\_fft.bin 8 10 2

rel\_rms\_err = 2.7e-07, usecs = 20, k = 0

rel\_rms\_err = 2.7e-07, usecs = 18, k = 1

Performance counter stats for './hello\_fft.bin 8 10 2':

35.376000 task-clock (msec) # 0.814 CPUs utilized

11 context-switches # 0.311 K/sec

0 cpu-migrations # 0.000 K/sec

97 page-faults # 0.003 M/sec

1,315,035 cycles # 0.037 GHz

4,507,197 stalled-cycles-frontend # 342.74% frontend cycles idle

441,497 stalled-cycles-backend # 33.57% backend cycles idle

4,967,122 instructions # 3.78 insns per cycle

# 0.91 stalled cycles per insn [59.55%]

942,058 branches # 26.630 M/sec [39.51%]

40,161 branch-misses # 4.26% of all branches [16.84%]

<not counted> L1-dcache-loads

<not counted> L1-dcache-load-misses

<not supported> LLC-loads

<not supported> LLC-load-misses

0.043437430 seconds time elapsed

$ sudo perf\_3.16 stat -d ./hello\_fft.bin 8 1 2

rel\_rms\_err = 3.3e-07, usecs = 53, k = 0

rel\_rms\_err = 3.3e-07, usecs = 34, k = 1

Performance counter stats for './hello\_fft.bin 8 1 2':

25.234000 task-clock (msec) # 0.741 CPUs utilized

17 context-switches # 0.674 K/sec

0 cpu-migrations # 0.000 K/sec

91 page-faults # 0.004 M/sec

3,416,788 cycles # 0.135 GHz

5,496,335 stalled-cycles-frontend # 160.86% frontend cycles idle

459,694 stalled-cycles-backend # 13.45% backend cycles idle

4,608,842 instructions # 1.35 insns per cycle

# 1.19 stalled cycles per insn [42.36%]

787,847 branches # 31.222 M/sec [18.20%]

<not counted> branch-misses

<not counted> L1-dcache-loads

<not counted> L1-dcache-load-misses

<not supported> LLC-loads

<not supported> LLC-load-misses

0.034069553 seconds time elapsed

$ sudo perf\_3.16 stat -d ./hello\_fft.bin 8 50 2

rel\_rms\_err = 2.5e-07, usecs = 18, k = 0

rel\_rms\_err = 2.5e-07, usecs = 18, k = 1

Performance counter stats for './hello\_fft.bin 8 50 2':

85.593000 task-clock (msec) # 0.905 CPUs utilized

13 context-switches # 0.152 K/sec

0 cpu-migrations # 0.000 K/sec

141 page-faults # 0.002 M/sec

59,446,895 cycles # 0.695 GHz [33.73%]

18,211,275 stalled-cycles-frontend # 30.63% frontend cycles idle [44.18%]

447,559 stalled-cycles-backend # 0.75% backend cycles idle

9,224,249 instructions # 0.16 insns per cycle

# 1.97 stalled cycles per insn [43.85%]

1,173,759 branches # 13.713 M/sec [33.63%]

72,115 branch-misses # 6.14% of all branches [28.20%]

82,373 L1-dcache-loads # 0.962 M/sec [27.02%]

58,372 L1-dcache-load-misses # 70.86% of all L1-dcache hits [26.00%]

<not supported> LLC-loads

<not supported> LLC-load-misses

0.094569758 seconds time elapsed

## Conclusion:

Rel\_rms\_err, time,

Bigger job size, bigger rel\_rms\_err.

More batches, less rel\_rms\_err.

More batches, less compute time for each task.

## FFT

# CPU part:

Numpy

Measure python 2.7

## Reference: [A guide to analyzing Python performance](https://www.huyng.com/posts/python-performance-analysis)

By: Huy Nguyen

<https://www.huyng.com/posts/python-performance-analysis>   
(Link is no longer useful)  
<http://zqpythonic.qiniucdn.com/data/20170602154836/index.html>

### Coarse grain timing with time

Let’s begin by using a quick and dirty method of timing our code: the good old unix utility time.

$ time python yourprogram.py

real 0m1.028s

user 0m0.001s

sys 0m0.003s

The meaning between the three output measurements are detailed in this [stackoverflow article](http://stackoverflow.com/questions/556405/what-do-real-user-and-sys-mean-in-the-output-of-time1), but in short

* real - refers to the actual elasped time
* user - refers to the amount of cpu time spent outside of kernel
* sys - refers to the amount of cpu time spent inside kernel specific functions

You can get a sense of how many cpu cycles your program used up regardless of other programs running on the system by adding together the sys and user times.

If the sum of sys and user times is much less than real time, then you can guess that most your program’s performance issues are most likely related to IO waits.

### Fine grain timing with a timing context manager

Our next technique involves direct instrumentation of the code to get access to finer grain timing information. Here’s a small snippet I’ve found invaluable for making ad-hoc timing measurements:

timer.py

import time

class Timer(object):

def \_\_init\_\_(self, verbose=False):

self.verbose = verbose

def \_\_enter\_\_(self):

self.start = time.time()

return self

def \_\_exit\_\_(self, \*args):

self.end = time.time()

self.secs = self.end - self.start

self.msecs = self.secs \* 1000 # millisecs

if self.verbose:

print 'elapsed time: %f ms' % self.msecs

In order to use it, wrap blocks of code that you want to time with Python’s withkeyword and this Timer context manager. It will take care of starting the timer when your code block begins execution and stopping the timer when your code block ends.

Here’s an example use of the snippet:

from timer import Timer

from redis import Redis

rdb = Redis()

with Timer() as t:

rdb.lpush("foo", "bar")

print "=> elasped lpush: %s s" % t.secs

with Timer() as t:

rdb.lpop("foo")

print "=> elasped lpop: %s s" % t.secs

I’ll often log the outputs of these timers to a file in order to see how my program’s performance evolves over time.

## Perf measure results:

e.g. perf measure the test demo:

$ sudo perf\_3.16 stat -d python2.7 test1.py

hello

a sum of 1 to 9 is: 45

b list of array is: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

0.00572299957275

Performance counter stats for 'python2.7 test1.py':

220.890000 task-clock (msec) # 0.946 CPUs utilized

13 context-switches # 0.059 K/sec

0 cpu-migrations # 0.000 K/sec

527 page-faults # 0.002 M/sec

153,804,192 cycles # 0.696 GHz [36.35%]

59,483,571 stalled-cycles-frontend # 38.67% frontend cycles idle [34.87%]

1,980,583 stalled-cycles-backend # 1.29% backend cycles idle [44.04%]

25,577,199 instructions # 0.17 insns per cycle

# 2.33 stalled cycles per insn [31.79%]

4,450,862 branches # 20.150 M/sec [28.31%]

481,440 branch-misses # 10.82% of all branches [26.68%]

1,475,384 L1-dcache-loads # 6.679 M/sec [27.85%]

263,682 L1-dcache-load-misses # 17.87% of all L1-dcache hits [26.46%]

<not supported> LLC-loads

<not supported> LLC-load-misses

0.233527929 seconds time elapsed