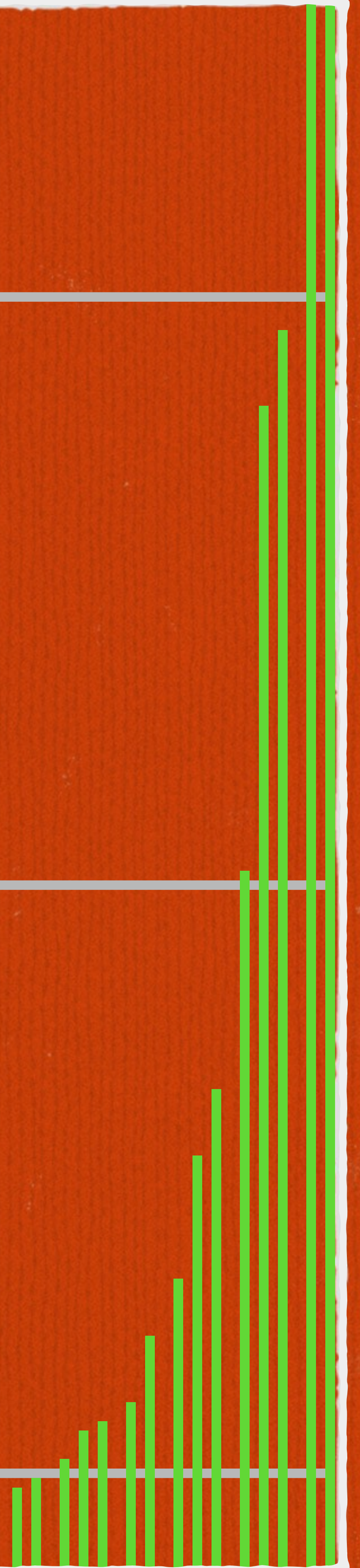
15.000.000

10.000.000

5.000.000



30 Years of CPS

The Rexx Clauses-per-second benchmark

René Jansen - RexxLA 2019

The Standard by MFC

```
CPS EXEC A1          At: "  /* Aver"..   2 changes

/*--------------------------------------------------------------------------*/
rexxcps=2.1    /* REXXCPS version; quotable only if code unchanged */

               /* Adjust these counts so run-time is about one second */
count=100      /* Repetition count */
averaging=100_ /* Averaging-over count */

tracevar='Off' /* Trace setting (for development use) */

signal on novalue
parse source  source  1 system .
parse version version


say '----- REXXCPS' rexxcps '-- Measuring REXX clauses/second -----'
say ' REXX version is:' version
say '       System is:' system
say '       Averaging:' averaging 'measures of' count 'iterations'

/* Calibrate for the empty do-loop */
empty=0
do i=1 to averaging
  call time 'R'
  do count; end
  empty=time('R')+empty
  end
empty=empty/averaging

noterm=(system='CMS'); if pos('O',tracevar)=1 then noterm=0
if noterm then do
  say 'Calibration (empty DO):' empty 'secs (average of' averaging')'
  say 'Spooling trace NOTERM'
  'CP SPOOL CON * START NOTERM'; 'CP CLOSE CON PUR'
    PF2  Opencl  PF3  Quit    PF4  Copy    PF5  Move    PF6  ?
    PF8  Down    PF9  =       PF10 Select  PF11 Splitj  PF12 Focus
```

- Measures Clauses Per Second
  - A clause is ~ a line of instructions
- History of measurements since 1989
- Product of analysis of 1000's of lines of real applications
- Multiplatform Classic Rexx

What does it do?

- Two loops
  - One for calibration of an (almost) empty loop
  - One that does the work
- Can specify # of averaging and measuring loops
- Total execution should be 1 sec elapsed time for dependable results

Calibrating the empty loop

```
 CPS      EXEC     B1  V 130  Trunc=130 Size=141 Line=0 Col=1 Alt=0
====>
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7
00000 * * * Top of File * * *
00001 --------------------  39  line(s) not displayed --------------------
00040 /* Calibrate for the empty do-loop */
00041 empty=0
00042 do i=1 to averaging
00043   call time 'R'
00044   do count; end
00045   empty=time('R')+empty
00046   end
00047 empty=empty/averaging
00048
00049 noterm=(system='CMS'); if pos('O',tracevar)=1 then noterm=0
00050 if noterm then do
00051   say 'Calibration (empty DO):' empty 'secs (average of' averaging')'
00052   say 'Spooling trace NOTERM'
00053   'CP SPOOL CON * START NOTERM'; 'CP CLOSE CON PUR'
00054   end
00055 --------------------  87  line(s) not displayed --------------------
00142 * * * End of File * * *
```

The timer loop

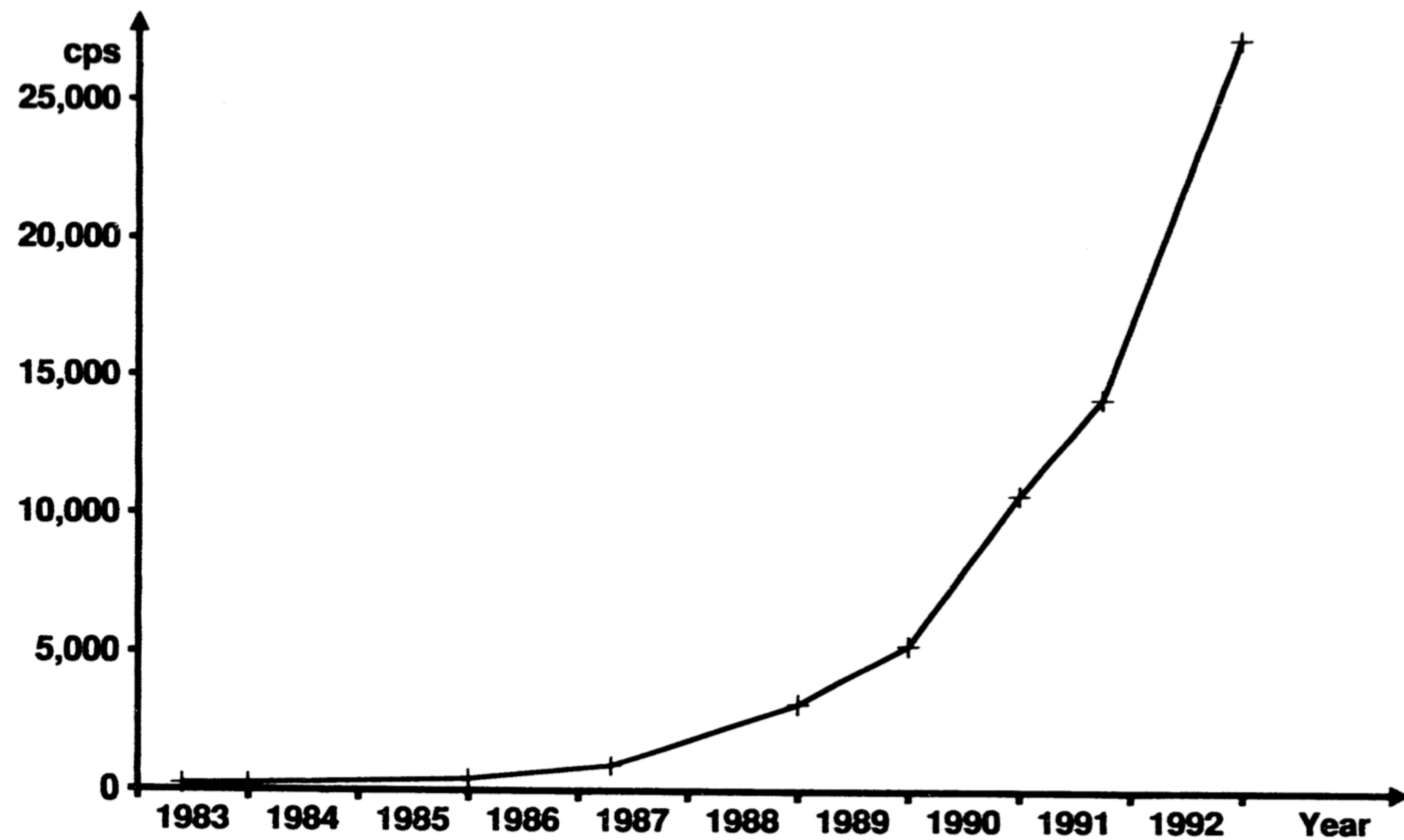
```
CPS      EXEC     B1  V 130  Trunc=130 Size=141 Line=0 Col=1
====>
       |...+....1....+....2....+....3....+....4....+....5....+.
00000 * * * Top of File * * *
00001 --------------------  55  line(s) not displayed --------
00056 /* Now the true timer loop .. average timing again */
00057 full=0
00058 do i=1 to averaging
00059   trace value tracevar
00060   call time 'R'
00061   do count;
00062     /* -----  This is first of the 1000 clauses ----- */
00063     flag=0; p0='b'
00064     do loop=1 to 14
00065       /* This is the "block" comment in loop */
00066       key1='Key Bee'
00067       acompound.key1.loop=substr(1234"5678",6,2)
00068       if flag=acompound.key1.loop then say 'Failed1'
00069       do j=1.1 to 2.2 by 1.1   /* executed 28 times */
00070         if j>acompound.key1.loop then say 'Failed2'
00071         if 17<length(j)-1        then say 'Failed3'
00072         if j='foobar'            then say 'Failed4'
00073         if substr(1234,1,1)=9    then say 'Failed5'
00074         if word(key1,1)='?'      then say 'Failed6'
00075         if j<5 then do   /* This path taken */
00076           acompound.key1.loop=acompound.key1.loop+1
00077           if j=2 then leave
00078           end
00079         iterate
00080         end /* j */
00081       avar.=1.0''loop
00082       select
00083         when flag='string' then say 'FailedS1'
00084         when avar.flag.2=0 then say 'FailedS2'
00085         when flag=5+99.7   then say 'FailedS3'
00086         when flag          then avar.1.2=avar.1.2*1.1
00087         when flag==0       then flag=0
00088         end
```

```
CPS      EXEC     B1  V 130  Trunc=130 Size=141 Line=88 Col=1 Alt=0
====>
       |...+....1....+....2....+....3....+....4....+....5....+....6....+....7.
00088         end
00089       if 1 then flag=1
00090       select
00091         when flag=='ring'  then say 'FailedT1'
00092         when avar.flag.3=0 then say 'FailedT2'
00093         when flag          then avar.1.2=avar.1.2*1.1
00094         when flag==0       then flag=1
00095         end
00096       parse value 'Foo Bar' with v1 +5 v2 .
00097       trace value trace(); address value address()
00098       call subroutine 'with' 2 'args', '(This is the second)'1''1
00099       rc='This is an awfully boring program'; parse var rc p1 (p0) p5
00100       rc='is an awfully boring program This'; parse var rc p2 (p0) p6
00101       rc='an awfully boring program This is'; parse var rc p3 (p0) p7
00102       rc='awfully boring program This is an'; parse var rc p4 (p0) p8
00103       end loop
00104     /* -----  This is last of the 1000 clauses ----- */
00105     end
00106   full=time('R')+full
00107   trace off
00108   end
00109 full=full/averaging
00110 --------------------  32  line(s) not displayed --------------------
00142 * * * End of File * * *
```

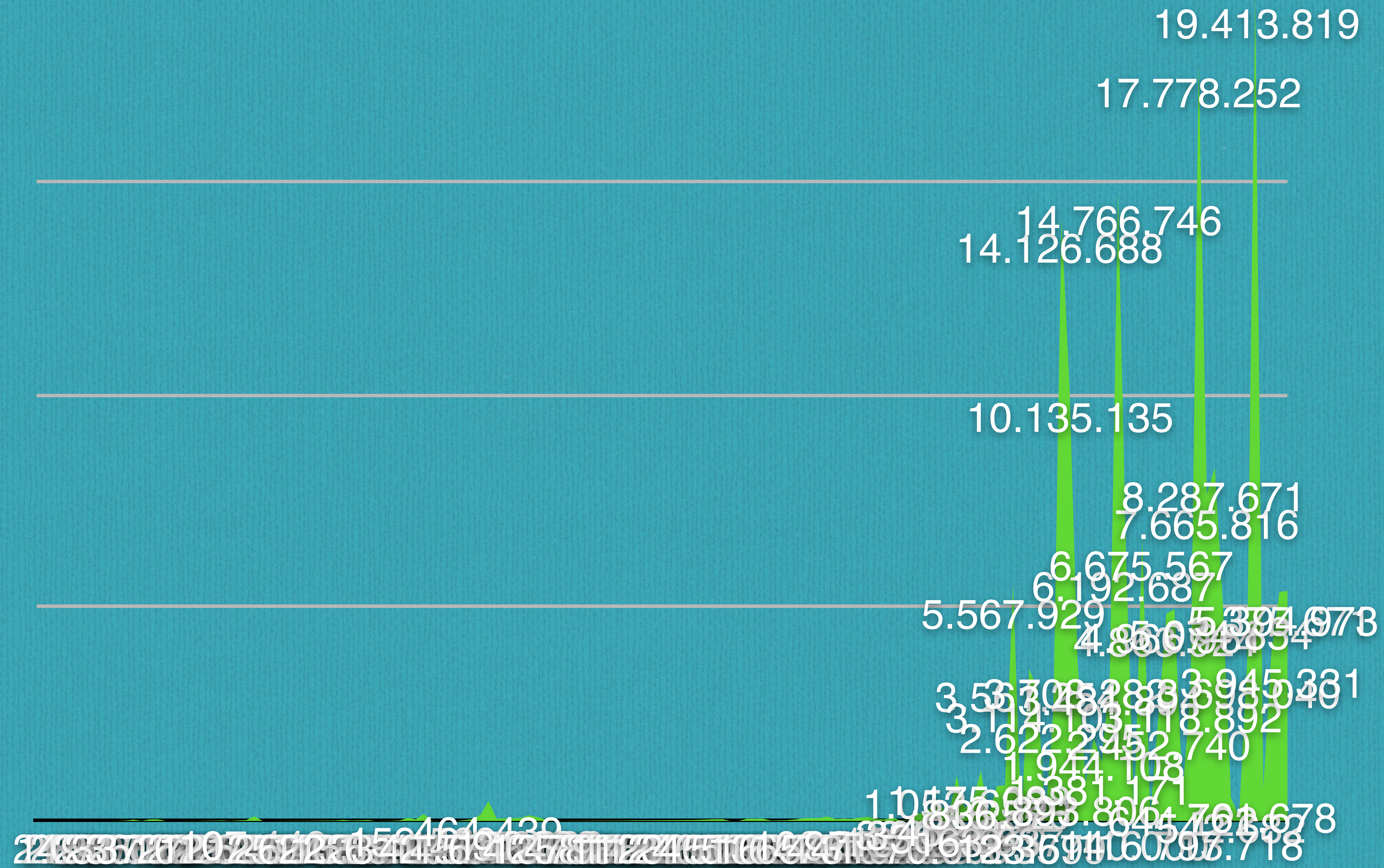
Hardware speed and REXX

As hardware speed increases, REXX is being used for a wider set of applications. Some informal figures:

- ✦ *x86* systems—now over 27,000 REXX clauses per second (486/66)
- ✦ RISC systems—over 42,000 REXX cps (same interpreter)
- ✦ Mainframe systems—over 90,000 REXX cps
- ✦ REXX Compiler/370—up to 465,000 REXX cps



WE APPEAR TO HAVE NUMBERS STARTING 1983


20.000.000
15.000.000
10.000.000
5.000.000
0
19.413.819
17.778.252
14.766.746
14.126.688
10.135.135
8.287.671
7.665.816
6.675.567
6.192.687
5.567.929
1992
2015

- But which is the ‘standard’ standard?
- Every interpreter delivers one
- All ever so slightly different
    - Brexx and ooRexx cannot run each other’s rexxcps
    - When fixed, they run their own cps faster than the other’s
- Canonical version on speleotrove.com

Influence of compiler options

Interpreter	Version	Architecture	OS	Virtualization	Compiler	Compiler Version	Compiler options	CPS score	Hardware	CPU	Date	Measures/ Iterations
brexx	2.1.9	s390x	Red Hat Enterprise Linux Server 7.5 (Maipo)	z/VM	gcc	4.8.5	"-O3 fno-stack-protector	6.961.707	IBM LinuxOne Rockhopper (Z13s)		2019-07-04	100/100
brexx	2.1.9	s390x	Red Hat Enterprise Linux Server 7.5 (Maipo)	z/VM	gcc	4.8.5	+ -mzarch	8.502.614	IBM LinuxOne Rockhopper (Z13s)		2019-07-04	100/100
brexx	2.1.9	s390x	Red Hat Enterprise Linux Server 7.5 (Maipo)	z/VM	gcc	4.8.5	+ "-m-hard-float" "-m-hard-dfp"	8.516.123	IBM LinuxOne Rockhopper (Z13s)		2019-07-04	100/100
brexx	2.1.9	s390x	Red Hat Enterprise Linux Server 7.5 (Maipo)	z/VM	gcc	4.8.5	+ "-march=z13"	8.567.324	IBM LinuxOne Rockhopper (Z13s)		2019-07-04	100/100
brexx	2.1.9	s390x	Red Hat Enterprise Linux Server 7.5 (Maipo)	z/VM	gcc	4.8.5	static executable	8.827.065	IBM LinuxOne Rockhopper (Z13s)		2019-07-04	100/100
REXX370 4.02 01 Dec 1998	4.02	z/Arch	CMS 6.4.0	z/VM				3.246.319	IBM LinuxOne Rockhopper (Z13s)		2018-11-15	100/100
brexx	2.1.9	x86_64	Ubuntu 18.04.2 LTS	no	gcc	7.4.0		19.620.712	Dell Latitude 5480	Intel Corei7-600U	2019-07-04	100/100

ARCHLVL

FP/DFP

Current numbers on x86_64 (Intel I9)

BREXX	21104425
ooRexx	13533448
Regina	8238630
Jaxx	9278788

Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
CPU(s): 4
On-line CPU(s) list: 0-3
Thread(s) per core: 2
Core(s) per socket: 2
Socket(s): 1
NUMA node(s): 1
Vendor ID: GenuineIntel
CPU family: 6
Model: 142
Model name: Intel(R) Core(TM) i7-7567U CPU @ 3.50GHz
Stepping: 9
CPU MHz: 1000.053
CPU max MHz: 4000.0000
CPU min MHz: 400.0000
BogoMIPS: 7008.00
Virtualization: VT-x
L1d cache: 32K
L1i cache: 32K
L2 cache: 256K
L3 cache: 4096K
NUMA node0 CPU(s): 0-3
Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_
fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_
ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust
xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp

Current numbers on ARM 64 (aarch64)

BREXX	2648138
ooRexx	1831838
Regina	2062977

```
rvjansen@jetson:~/data/brexx/src$ lscpu
Architecture:       aarch64
Byte Order:         Little Endian
CPU(s):             4
On-line CPU(s) list: 0-3
Thread(s) per core:  1
Core(s) per socket:  4
Socket(s):          1
Vendor ID:          ARM
Model:              1
Model name:         Cortex-A57
Stepping:           r1p1
CPU max MHz:        1428.0000
CPU min MHz:        102.0000
BogoMIPS:           38.40
L1d cache:          32K
L1i cache:          48K
L2 cache:           2048K
Flags:              fp asimd evtstrm aes pmull sha1 sha2 crc32
```

Current numbers on ARM 32 (armv7l)

BREXX	2816483
ooRexx	1914118
Regina	2043062

```
pi@kleene:~/apps $ lscpu
Architecture:        armv7l
Byte Order:          Little Endian
CPU(s):              4
On-line CPU(s) list: 0-3
Thread(s) per core:  1
Core(s) per socket:  4
Socket(s):           1
Vendor ID:           ARM
Model:               3
Model name:          Cortex-A72
Stepping:            r0p3
CPU max MHz:         1500.0000
CPU min MHz:         600.0000
BogoMIPS:            108.00
Flags:               half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva id
```

Why is ooRexx slower?

- Interpreter api
- Stems are slower than array objects
-

Why is BREXX faster?

- Limited numeric precision
- Not an exact implementation of the standard

CRX

- Written in Intel assembler (masm) for DOS and Windows, very fast
- Its purpose was to verify the ANSI (ISO) standard
- Parts of it generated from the grammar in that document
- Made with knowledge of Intel processor cache and pipelines

Parallel Rexx

- Maybe the future
- Nvidia and Intel have competing ‘AI’ processors
- CUDA in video cards exists for several years now, CUDA C++ can - with your assistance, that is - parallellize selected loops to execute hundreds times faster, consistent with the number of GPU’s available
- To be faster we cannot really trust the CPUs itself
- Experiments would be very interesting

NetRexx CPS

There never was one

Or was there?

NetRexx

Links

- Learn more ...
- NetRexx Tools
- NetRexx Forum
- Rexx Language Association

Welcome to the NetRexx Programming Language!

NetRexx is a general-purpose programming language inspired by two very different progra
computers. In this respect it follows Rexx closely, with many of the concepts and most of th
Rexx. From Java it derives static typing, binary arithmetic, the object model, and exception
Originally a product from the IBM Hursley Software Lab, NetRexx has always been free so
alternative language for the Java Virtual Machine (JVM).

Why NetRexx?

- NetRexx makes programming easy, and fun again
- The Rexx type, combining numeric and string processing in one class
- Unlimited precision arithmetic built into the language
- Interpret your code or compile to JVM .class files
- Lightning fast performance compared to other JVM scripting languages
- Seamless integration to all JVM libraries
- Built-in luxurious parsing and tracing

Hursley Labs

> Hursley, located near Winchester in the UK, is the place where many famous product
> to the qtime program, one of the first-ever Rexx programs, dating from 1979. This is t
> NetRexx listens to both *center* and *centre* method spellings.

MFC on NetRexx

> The seamless integration of types into what was previously an essentially typeless la
> strong typing while preserving the ease of use and speed of development that Rexx p
> capabilities to the Rexx language is a single language that has both the Rexx strengt
> strengths of robustness, good efficiency, portability, and security for application devel
> -- Mike Cowlishaw

rexxcps.nrx

- From 1996, by Mike Cowlishaw, but never released until now
- Optimizing compilers optimize a lot of code away, must make sense of the numbers
- In NetRexx 3.08, there is an experimental version released
  - Uses Nanotime instead of millisecs
  - Currently evaluated on different architectures to see what actually is executed

- NetRexx programs have a complex runtime
- NetRexx is translated to Java source, which is compiled to bytecode, which is interpreted (sometimes) but mostly compiled just-in-time to native code
- Optimization takes place on different levels, each can throw out code
- Reaches extremely high numbers

Current rexxcps.nrx

Aarch64	2,407E+15	2_407_464_380_676
IA 86_64 I9	1,1489E+16	11_488_524_444_376
IA 86_64 I7	9,861E+15	9_861_203_229_267

How do we see what is executed?

List the generated java
List the generated bytecode
List the generated assembler

```
File Edit Options Buffers Tools Index NetRexx Help
say '          Averaging:' averaging 'measures of' count 'iteratio

/* ----- Calibrate for the empty loop ----- */
empty=long 0
loop for averaging
  start=System.nanoTime()
  loop for count; end
    empty=empty+System.nanoTime()-start
  end
  fempty=empty/averaging

  /* ----- Now the true timer loop .. average timing again ----- */
  full=long 0
  loop for averaging
    start=System.nanoTime()
    loop for count
      /* -----  This is first of the 1000 clauses ----- */
      flag=0; p0='b'
      loop lvar=1 to 14
        /* This is the "block" comment in lvar */
        key1='Key Bee'; acompound=''
        acompound[key1,lvar]=(1234"5678").substr(6,2)
        if flag=acompound[key1,lvar] then say 'Failed1'
        loop j=1.1 to 2.2 by 1.1   /* executed 28 times */
          if j>acompound[key1,lvar] then say 'Failed2'
          if 17<key1.length-1       then say 'Failed3'
          if j='foobar'             then say 'Failed4'
          if key1.substr(1,1)=9     then say 'Failed5'
          if key1.word(1)='?'       then say 'Failed6'
          if j<5 then do   /* This path taken */
            acompound[key1,lvar]=acompound[key1,lvar]+1
            if j=2 then leave /* never */
          end
          iterate
        end /* j */
        avar=1.0''lvar
        select
```

.nrx yields .java

We can verify a 1:1 relationship between the nrx and the java sourcecode

```
ssh
e Edit Options Buffers Tools Java Help
{int $5=count.OpPlus($1).toint();for(;$5>0;$5--){
 }
}
empty=new netrexx.lang.Rexx(empty).OpAdd($1,new netrexx.lang.Rexx(java.lang.
(start)).tolong();
}

empty=new netrexx.lang.Rexx(empty).OpDiv($1,averaging);
ull=new netrexx.lang.Rexx(0).tolong();
int $6=averaging.OpPlus($1).toint();for(;$6>0;$6--){
start=java.lang.System.nanoTime();
{int $7=count.OpPlus($1).toint();for(;$7>0;$7--){
 flag=new netrexx.lang.Rexx((byte)0);
 p0=new netrexx.lang.Rexx('b');
 {lvar=new netrexx.lang.Rexx((byte)1);lvar:for(;lvar.OpLtEq($1,$07);lvar=lva
  key1=netrexx.lang.Rexx.toRexx("Key Bee");
  acompound=netrexx.lang.Rexx.toRexx("");
  acompound.getnode(key1).leaf.getnode(lvar).leaf=($08.OpCc($1,$09)).substr(
xx((byte)2));
  if (flag.OpEq($1,acompound.getnode(key1).leaf.getnode(lvar).leaf))
   netrexx.lang.RexxIO.Say("Failed1");
  {netrexx.lang.Rexx $8=$010;j=$010.OpPlus($1);j:for(;j.OpLtEq($1,$011);j=j.
   if (j.OpGt($1,acompound.getnode(key1).leaf.getnode(lvar).leaf))
    netrexx.lang.RexxIO.Say("Failed2");
   if ($013.OpLt($1,(key1.length()).OpSub($1,$012)))
    netrexx.lang.RexxIO.Say("Failed3");
   if (j.OpEq($1,$014))
    netrexx.lang.RexxIO.Say("Failed4");
   if ((key1.substr(new netrexx.lang.Rexx((byte)1),new netrexx.lang.Rexx((by
    netrexx.lang.RexxIO.Say("Failed5");
   if ((key1.word(new netrexx.lang.Rexx((byte)1))).OpEq($1,$016))
    netrexx.lang.RexxIO.Say("Failed6");
   if (j.OpLt($1,$017))
    {
     acompound.getnode(key1).leaf.getnode(lvar).leaf=(acompound.getnode(key1
     if (j.OpEq($1,$018))
      break j;
-:----F1  rexxcps.java    40% L107   (Java/l Abbrev) ------------------------
```

.java yields .class

We can verify that all bytecode for the benchmark has been generated

```
     540: new           #1                  // class netrexx/lang/F
     543: dup
     544: iconst_2
     545: invokespecial #7                  // Method netrexx/lang/
     548: invokevirtual #39                 // Method netrexx/lang/
x/lang/Rexx;
     551: putfield      #35                 // Field netrexx/lang/F
     554: aload         15
     556: getstatic     #11                 // Field $1:Lnetrexx/la
     559: aload         19
     561: aload         18
     563: invokevirtual #34                 // Method netrexx/lang/
     566: getfield      #35                 // Field netrexx/lang/F
     569: aload         17
     571: invokevirtual #34                 // Method netrexx/lang/
     574: getfield      #35                 // Field netrexx/lang/F
     577: invokevirtual #40                 // Method netrexx/lang/
     580: ifeq          589
     583: ldc           #41                 // String Failed1
     585: invokestatic  #42                 // Method netrexx/lang/
     588: pop
     589: getstatic     #43                 // Field $010:Lnetrexx/
     592: astore        39
     594: getstatic     #43                 // Field $010:Lnetrexx/
     597: getstatic     #11                 // Field $1:Lnetrexx/la
     600: invokevirtual #22                 // Method netrexx/lang/
     603: astore        20
     605: aload         20
     607: getstatic     #11                 // Field $1:Lnetrexx/la
     610: getstatic     #44                 // Field $011:Lnetrexx/
     613: invokevirtual #31                 // Method netrexx/lang/
     616: ifeq          867
     619: aload         20
     621: getstatic     #11                 // Field $1:Lnetrexx/la
     624: aload         19
     626: aload         18
     628: invokevirtual #34                 // Method netrexx/lang/
-UU-:----F1  rexxcps.b      19% L344    (Fundamental) -------------
```

The Empty Averaging Loop in Java

ssh ⌥⌘1

```
File Edit Options Buffers Tools Java Help
  count=new netrexx.lang.Rexx((byte)100);
  averaging=new netrexx.lang.Rexx((byte)100);
  rexxcps=netrexx.lang.Rexx.toRexx("2.1n");
  netrexx.lang.RexxIO.Say((netrexx.lang.Rexx.toRexx("----- REXXCPS").OpCcblank($1,rexxcps)).OpCcblank($1,netrexx.lang.Rexx.toR\
exx("-- Measuring NetRexx clauses/second -----")));
  netrexx.lang.RexxIO.Say(netrexx.lang.Rexx.toRexx(" NetRexx version is:").OpCcblank($1,version));
  netrexx.lang.RexxIO.Say(netrexx.lang.Rexx.toRexx("          System is:").OpCcblank($1,opsystem));
  netrexx.lang.RexxIO.Say(((netrexx.lang.Rexx.toRexx("          Averaging:").OpCcblank($1,averaging)).OpCcblank($1,netrexx.la\
ng.Rexx.toRexx("measures of"))).OpCcblank($1,count)).OpCcblank($1,$06));
  empty=new netrexx.lang.Rexx(0).tolong();
  {int $4=averaging.OpPlus($1).toint();for(;$4>0;$4--){
   start=java.lang.System.nanoTime();
   {int $5=count.OpPlus($1).toint();for(;$5>0;$5--){
    }
   }
   empty=new netrexx.lang.Rexx(empty).OpAdd($1,new netrexx.lang.Rexx(java.lang.System.nanoTime())).OpSub($1,new netrexx.lang.R\
exx(start)).tolong();
   }
  }
  fempty=new netrexx.lang.Rexx(empty).OpDiv($1,averaging);
  full=new netrexx.lang.Rexx(0).tolong();
  {int $6=averaging.OpPlus($1).toint();for(;$6>0;$6--){
   start=java.lang.System.nanoTime();
   {int $7=count.OpPlus($1).toint();for(;$7>0;$7--){
    flag=new netrexx.lang.Rexx((byte)0);
    p0=new netrexx.lang.Rexx('b');
    {lvar=new netrexx.lang.Rexx((byte)1);lvar:for(;lvar.OpLtEq($1,$07);lvar=lvar.OpAdd($1,new netrexx.lang.Rexx(1))){
     key1=netrexx.lang.Rexx.toRexx("Key Bee");
     acompound=netrexx.lang.Rexx.toRexx("");
     acompound.getnode(key1).leaf.getnode(lvar).leaf=($08.OpCc($1,$09)).substr(new netrexx.lang.Rexx((byte)6),new netrexx.lang\
.Rexx((byte)2));
     if (flag.OpEq($1,acompound.getnode(key1).leaf.getnode(lvar).leaf))
      netrexx.lang.RexxIO.Say("Failed1");
     {netrexx.lang.Rexx $8=$010;j=$010.OpPlus($1);j:for(;j.OpLtEq($1,$011);j=j.OpAdd($1,$8)){
      if (j.OpGt($1,acompound.getnode(key1).leaf.getnode(lvar).leaf))
       netrexx.lang.RexxIO.Say("Failed2");
      if ($013.OpLt($1,(key1.length()).OpSub($1,$012)))
-UU-:----F1  rexxcps.java   33% L108   (Java/l Abbrev) ----------------------------------------------------------------
```

Generated Assembler

- With a special dll we can see what the HotSpot compiler generates as assembly
- We can also see and influence which parts will be compiled and which parts will be interpreted

Parallellism

- Rexxcps.nrx also is a single tasking program
- As java can also be parallellized, the future might hold change
- Pipelines in NetRexx, for example are already multitasked over all available processors

```
➜  test git:(master) ✗ time pipe 'literal aap noot mies | split | reverse | a: locate seim ? a: | console'
paa
toon
java org.netrexx.njpipes.pipes.runner   2.06s user 0.12s system 249% cpu 0.870 total
➜  test git:(master) ✗
```

Example of Parallelism

2.067 seconds are spent with 249% cpu, which makes for an elapsed time of 0.87 seconds