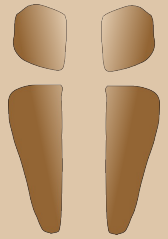


Parallelism: The JVM Rocks

Dr Russel Winder

It'z Interactive Ltd

russel@itzinteractive.com



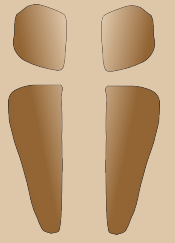
Aims and Objectives

- Investigate why the JVM is the platform of choice in a multicore world.
- Investigate various parallel program features on the JVM introduced by various languages:
 - Java, `java.util.concurrent`
 - Groovy, GParS
 - Scala
 - Clojure
 - Fortress
 - X10
- Have fun (whilst learning something).

g/GParallelizer/s//GParS/p

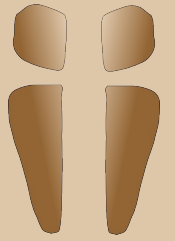


<http://gpars.codehaus.org>



Audience Aims and Objectives

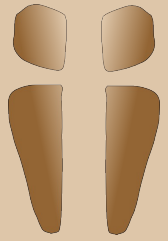




Structure of the Session

- Introduction.
- Do stuff.
- Exit stage (left | right).

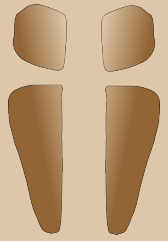
There *will* be significant dynamic binding of the session.



Protocol

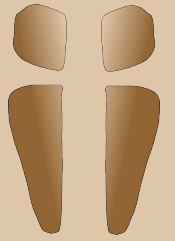
- Some slides, to kick things off.
- Some programs to really demonstrate things.
- *NB Interaction between audience and presenter is mandatory; a necessity not an option.*

We reserve the right to (shelve\stash) for later a particular interaction if it goes on longer than seems appropriate.



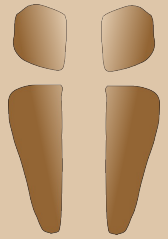
Historical Points

- Shared-memory multi-threading for applications programming is an aberration:
 - Consequences of operating systems handling of concurrency have been imposed on all programmers.
- Cooperating processes is where sanity is:
 - Operating system interprocess communication was slow, hence threads, but this lead directly to the shared-memory, multi-threading quagmire.
 - Erlang shows that processes and message passing can do the job properly even after the “mainstream” had rejected process-based working..



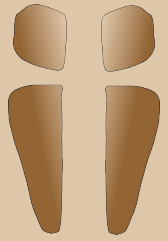
JVM

- Aka Java Virtual Machine.
- The bytecode interpreter designed for executing Java:
 - Originally designed for a distributed code context.
 - Code distributed over the Internet.
 - *You will live in a sandbox.*



The JVM . . .

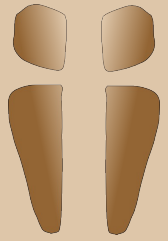
- . . . “commoditizes” platform:
 - Windows
 - Linux
 - Mac OS X
 - Solaris
- } We don't have to care about this factor when using the JVM.



Java

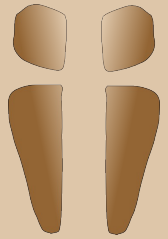
- Has a history – but we don't really care about it.
- Handles concurrency by using threads:
 - Threads operate in a shared-memory context so synchronization is required:
 - Object locks.
 - Synchronized statements.
 - Monitors.

The need for explicit synchronization is the root of all evil.



Concurrency

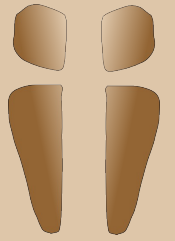
- The problem with threads is shared memory:
 - Without writeable shared memory there is no need for synchronization.
- Why impose a 1960s, operating system driven view of concurrency management on applications programmers?



Parallelism is . . .

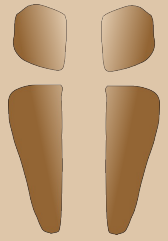
- . . . a subset of concurrency:
 - Parallelism is where elements of a concurrent system execute at the same time physically not just virtually.

The consequences of explicit synchronization are worse with parallelism than with concurrency.



Shared Memory is . . .

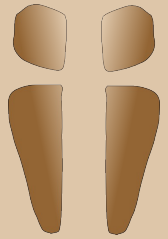
- . . . anathema to parallelism.
- . . . anathema to concurrency.
- . . . anathema to modern applications programming.
- . . . anathema.



Partial Solution

- Use `java.util.concurrent`:
 - Provides many high-level tools.
 - Has many low-level tools.

If you are using the low-level tools then you are lost to the cause of quality application programming.



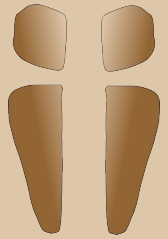
CSP is . . .

- . . . mathematics:

$$\text{VMS} = \mu X. (\text{in2p} \rightarrow (\text{chocolate} \rightarrow X | \text{out1p} \rightarrow \text{toffee} \rightarrow X) \\ | \text{in1p} \rightarrow (\text{toffee} \rightarrow X | \text{in1p} \rightarrow (\text{chocolate} \rightarrow X | \text{in1p} \rightarrow \text{STOP} \alpha X)))$$

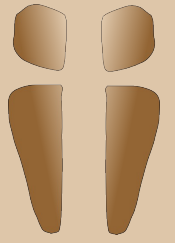
- . . . not scary since the mathematics can be hidden in an API, so it just becomes a programming tool.

JCSP will come in handy.

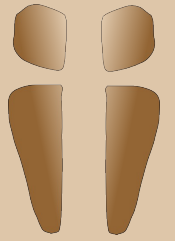


Example Problem

- Something small, so the code is small.
- Something not too “toy”.
- Something with good parallelism.
 - *Embarrassingly parallel* to allow checking of *scaling*.

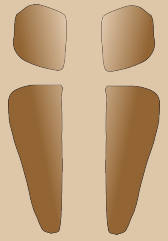


π



What is the Value of π ?

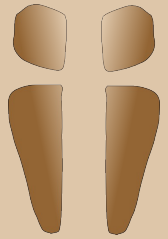
- The value of π is known exactly, it's π (obviously).



Approximating π

- What is its value represented as a floating point number?
 - We can only obtain an approximation.
 - A plethora of possible algorithms to choose from, a popular one is to employ the following integral equation.

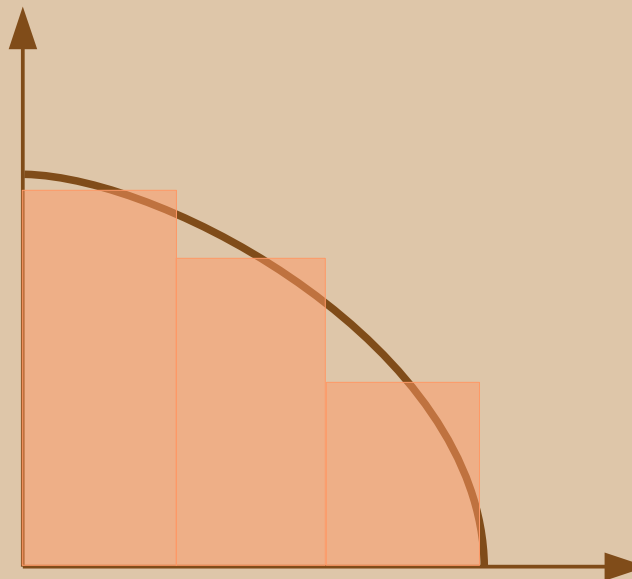
$$\frac{\pi}{4} = \int_0^1 \frac{1}{1+x^2} dx$$



One Possible Algorithm

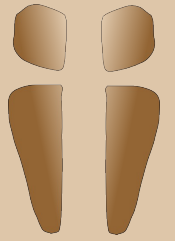
- Use quadrature to estimate the value of the integral – which is the area under the curve.

$$\pi = \frac{4}{n} \sum_{i=1}^n \frac{1}{1 + \left(\frac{i-0.5}{n}\right)^2}$$



Embarrassingly parallel.

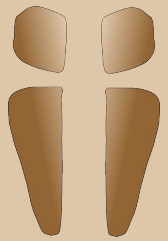
With $n = 3$ not much to do, but potentially lots of error.



The Machine

- Twin Xeon E5410 2.33GHz.
- 4GB memory.
- Ubuntu 10.04 Lucid Lynx AMD64.

*She's called Anglides,
do say hello.*

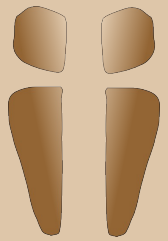


Java Sequential

Code

pi = 3.1415926535899708
iteration count = 1000000000
elapse = 18.4872549





Java Threads

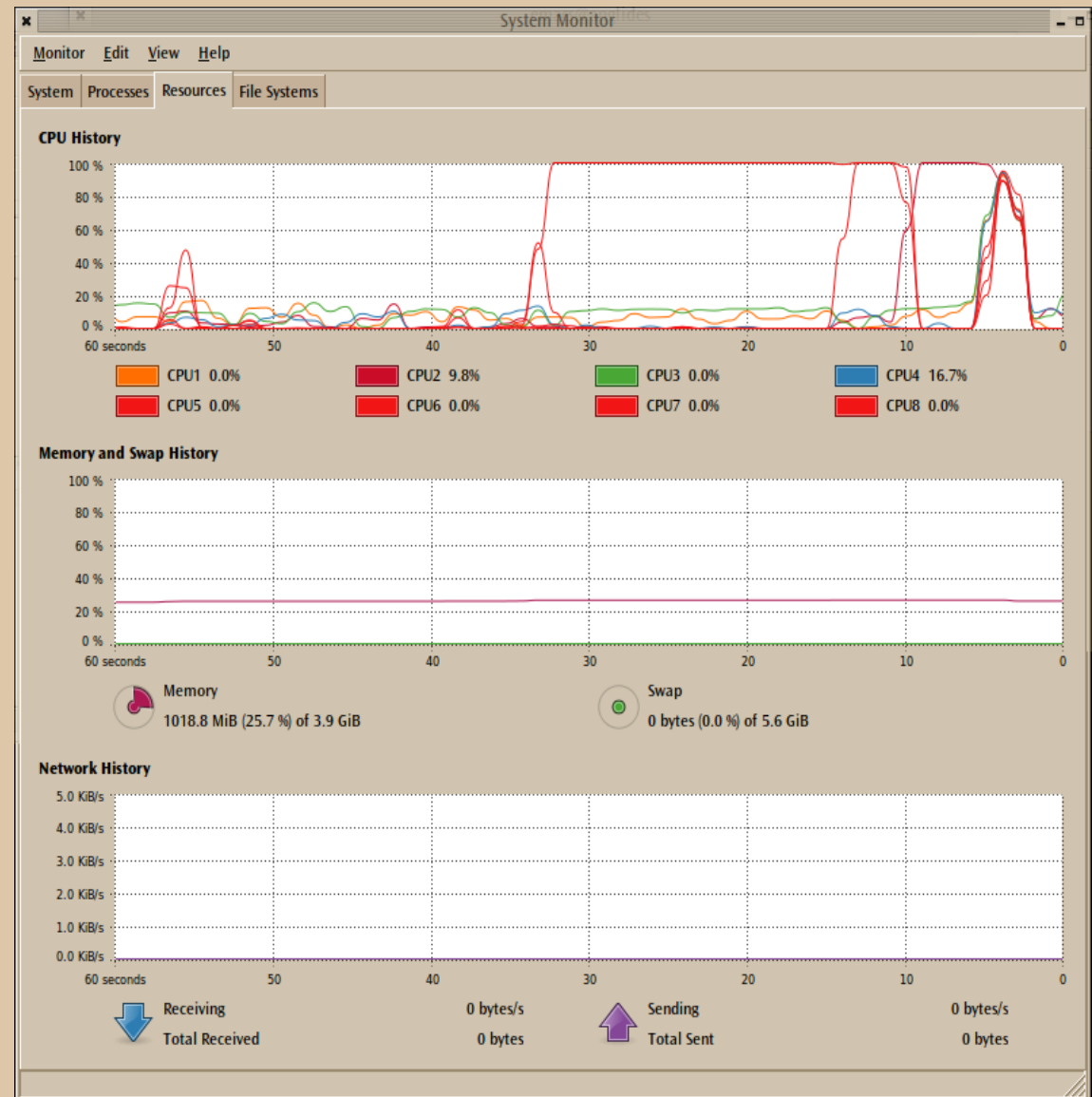
Code

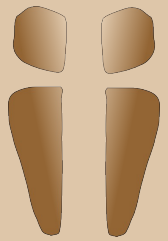
pi = 3.1415926535899708
iteration count = 1000000000
elapse = 18.495807275
processor count = 8
thread count = 1

pi = 3.141592653589901
iteration count = 1000000000
elapse = 9.262973638
processor count = 8
thread count = 2

pi = 3.141592653589769
iteration count = 1000000000
elapse = 1.190413043
processor count = 8
thread count = 8

pi = 3.141592653589758
iteration count = 1000000000
elapse = 1.197404584
processor count = 8
thread count = 32





Java Futures

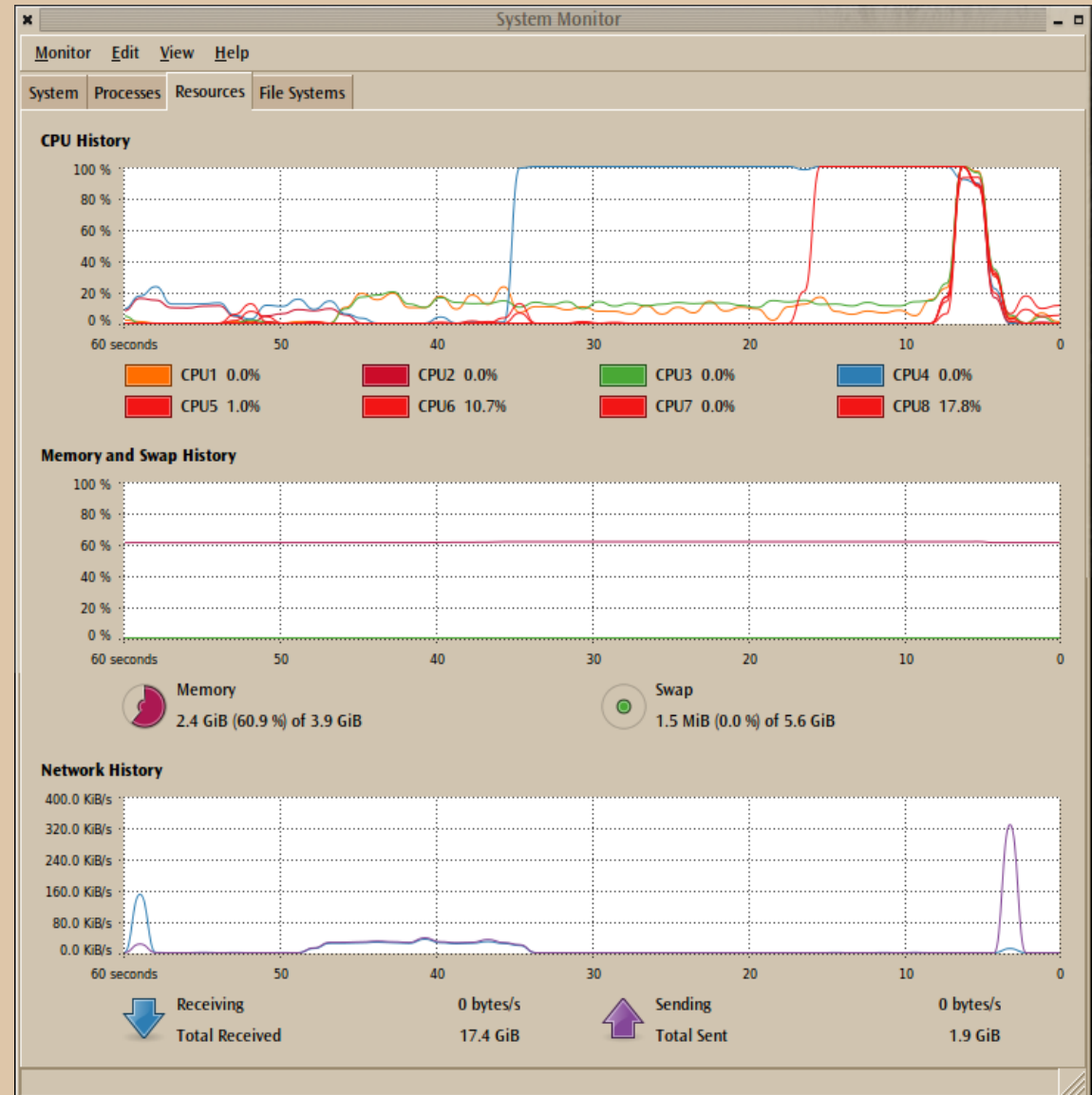
Code

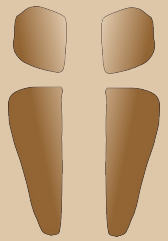
pi = 3.1415926535899708
iteration count = 1000000000
elapse = 18.522036642
processor count = 8
thread count = 1

pi = 3.141592653589901
iteration count = 1000000000
elapse = 9.242476655
processor count = 8
thread count = 2

pi = 3.1415926535897687
iteration count = 1000000000
elapse = 1.201076547
processor count = 8
thread count = 8

pi = 3.141592653589758
iteration count = 1000000000
elapse = 1.196214241
processor count = 8
thread count = 32





Java ForkJoin

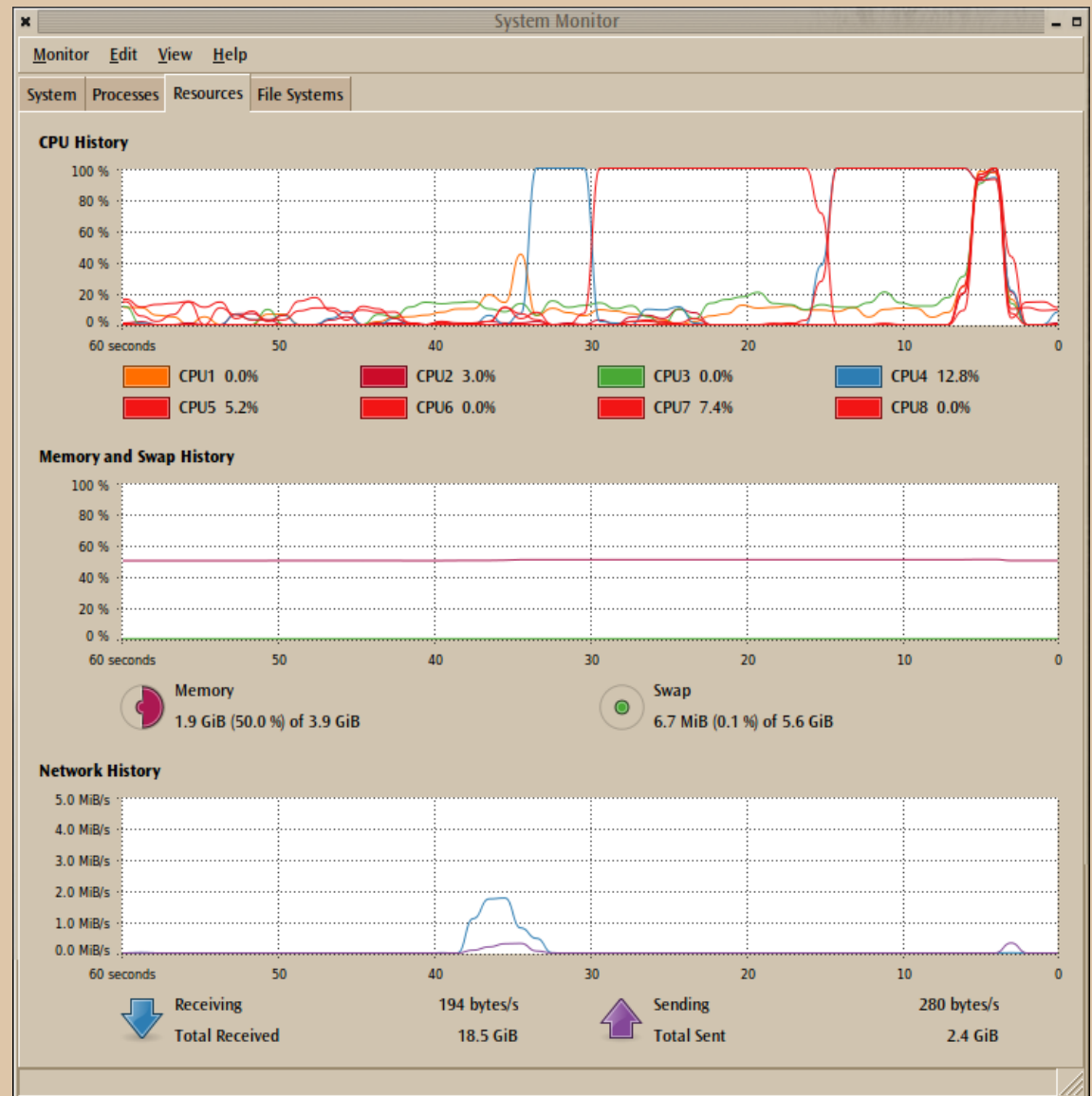
Code

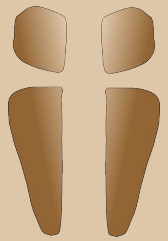
pi = 3.1415926535899708
iteration count = 1000000000
elapse = 18.565483486
processor count = 8
thread count = 1

pi = 3.141592653589901
iteration count = 1000000000
elapse = 9.243049998
processor count = 8
thread count = 2

pi = 3.1415926535897687
iteration count = 1000000000
elapse = 1.163065908
processor count = 8
thread count = 8

pi = 3.141592653589758
iteration count = 1000000000
elapse = 1.162013855
processor count = 8
thread count = 32





Java Parallel Array

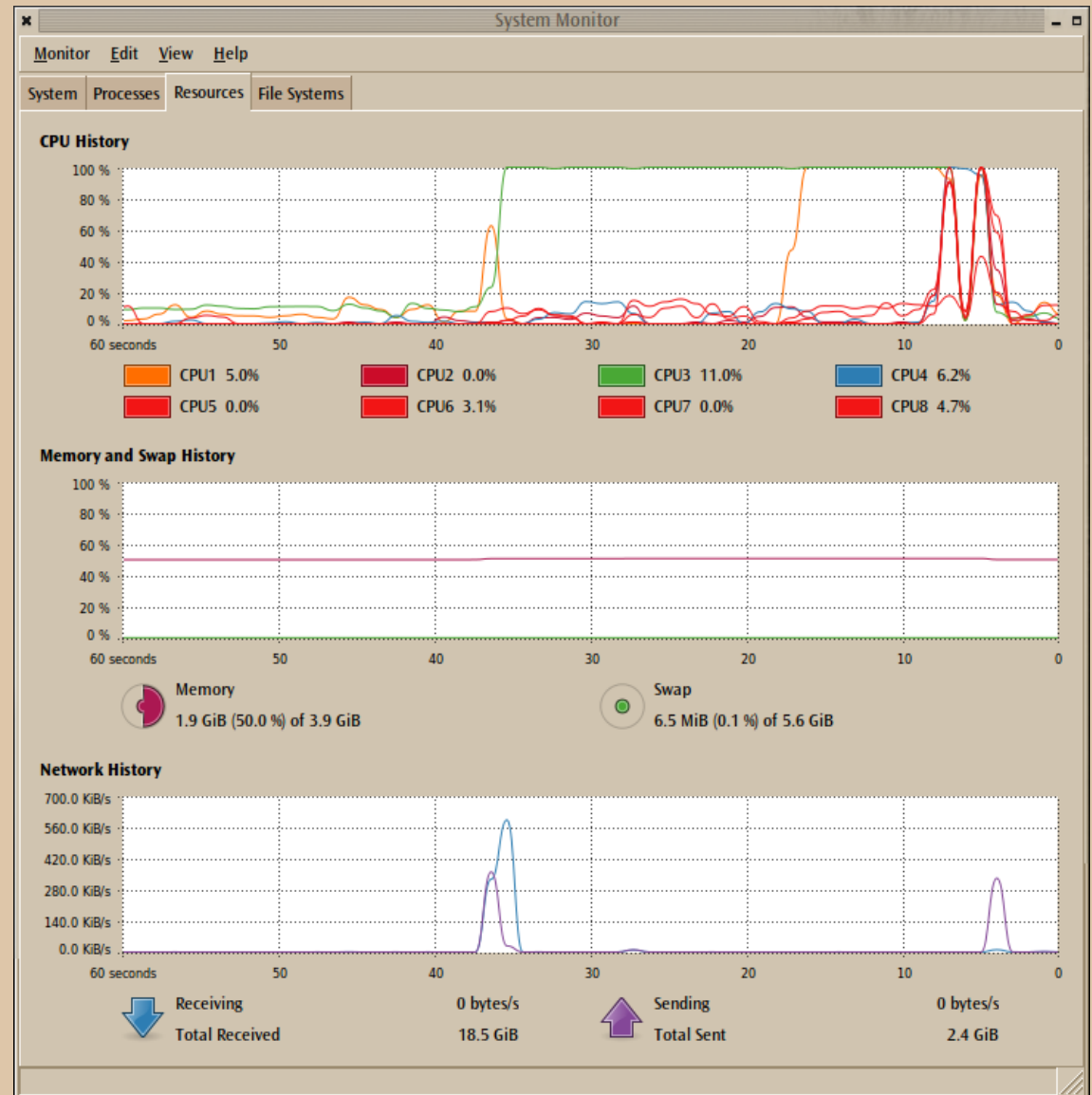
Code

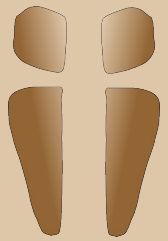
pi = 3.1415926535899708
iteration count = 1000000000
elapse = 18.542106742
processor count = 8
thread count = 1

pi = 3.141592653589901
iteration count = 1000000000
elapse = 9.244273314
processor count = 8
thread count = 2

pi = 3.1415926535897687
iteration count = 1000000000
elapse = 2.14923197
processor count = 8
thread count = 8

pi = 3.141592653589757
iteration count = 1000000000
elapse = 1.394784886
processor count = 8
thread count = 32





Java JCSP

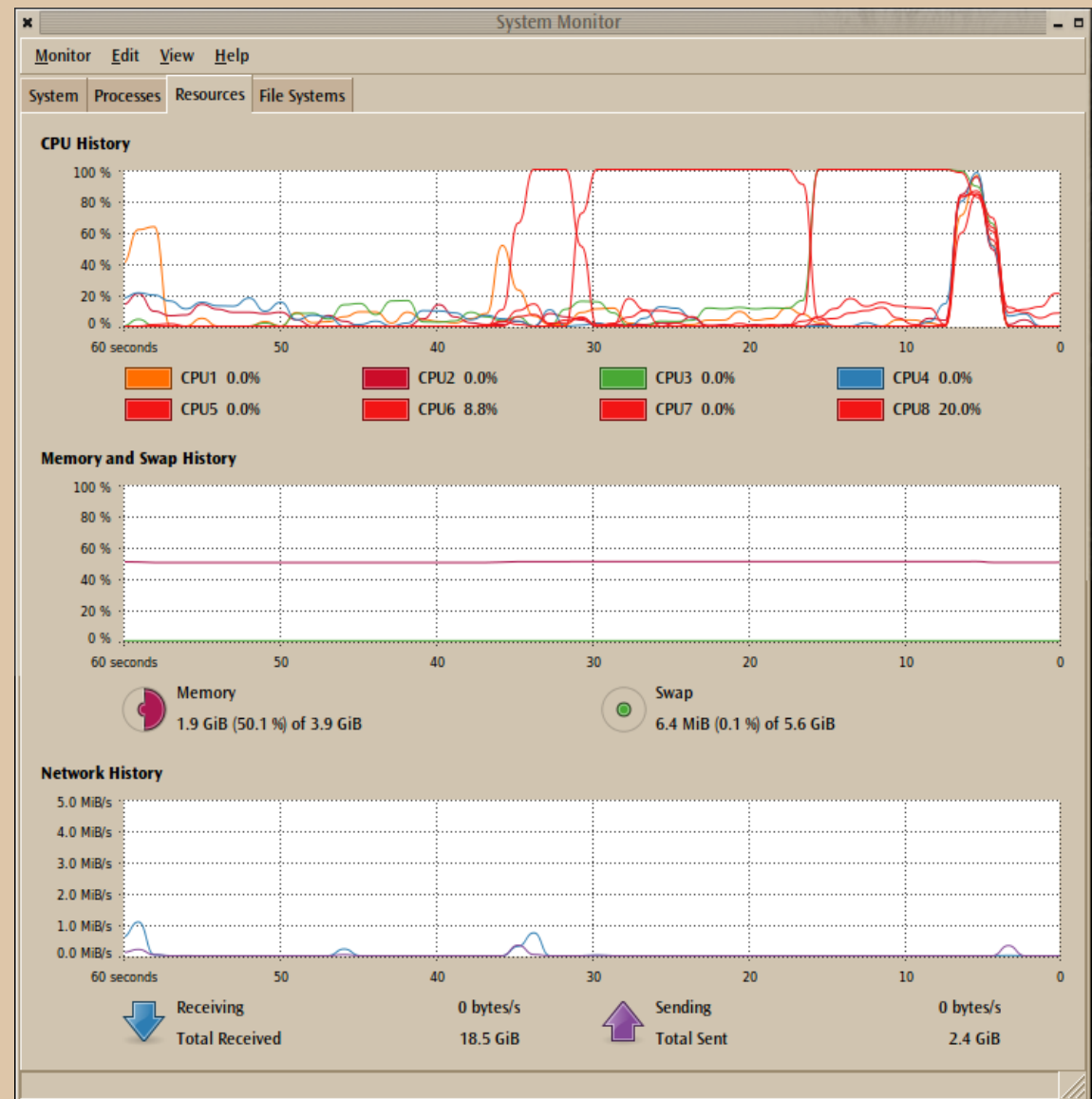
Code

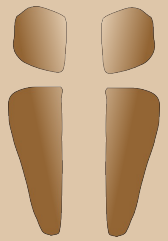
pi = 3.1415926535899708
iteration count = 1000000000
elapse = 18.570761732
processor count = 8
task count = 1

pi = 3.141592653589901
iteration count = 1000000000
elapse = 9.24588872
processor count = 8
task count = 2

pi = 3.1415926535897687
iteration count = 1000000000
elapse = 1.219945744
processor count = 8
task count = 8

pi = 3.141592653589758
iteration count = 1000000000
elapse = 1.20014776
processor count = 8
task count = 32





Functional Java

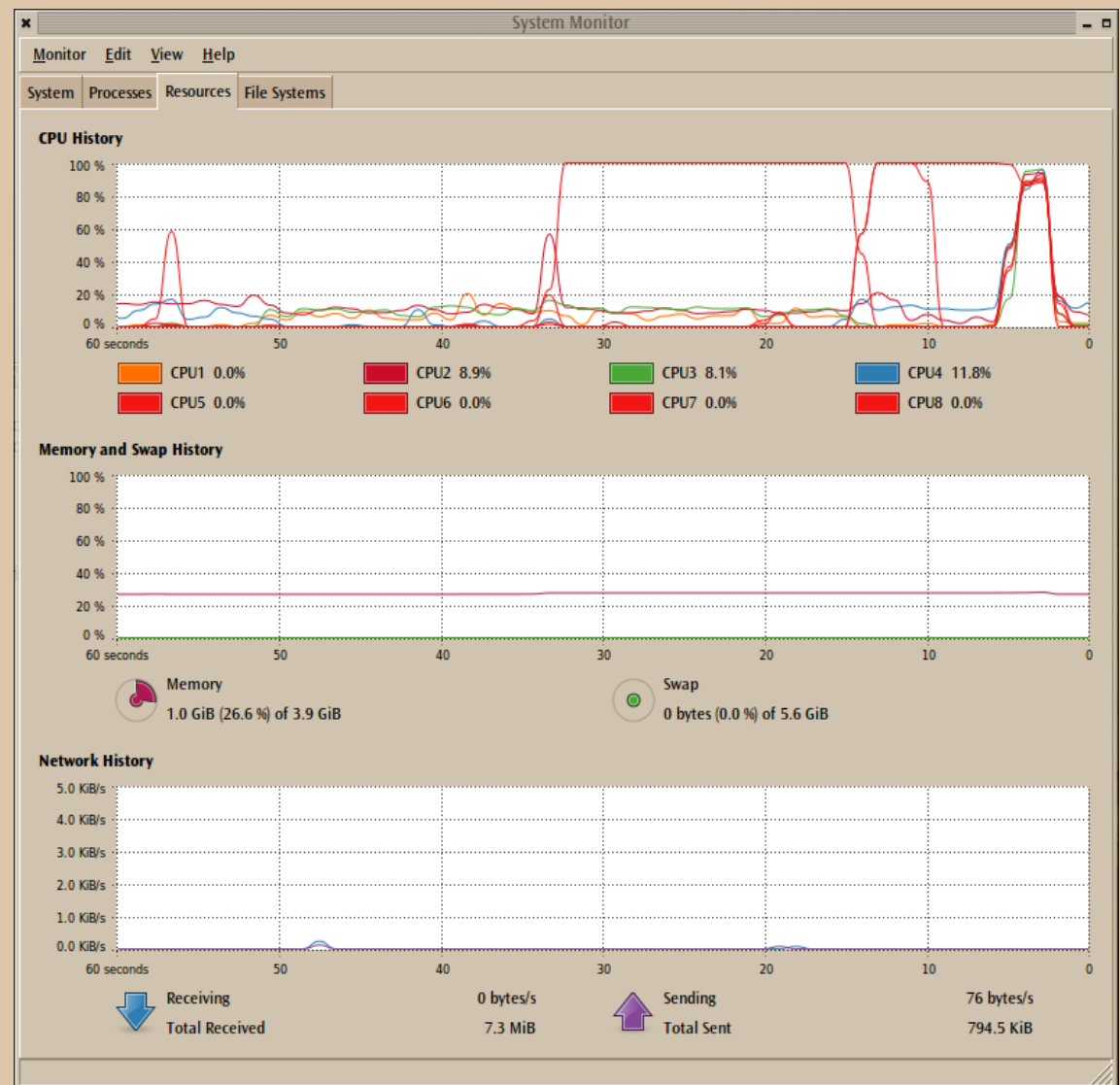
Code

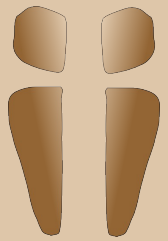
pi = 3.1415926535899708
iteration count = 1000000000
elapse = 18.632565146
processor count = 8
thread count = 1

pi = 3.141592653589901
iteration count = 1000000000
elapse = 9.248000676
processor count = 8
thread count = 2

pi = 3.1415926535897687
iteration count = 1000000000
elapse = 1.244986601
processor count = 8
thread count = 8

pi = 3.141592653589758
iteration count = 1000000000
elapse = 1.360103422
processor count = 8
thread count = 32

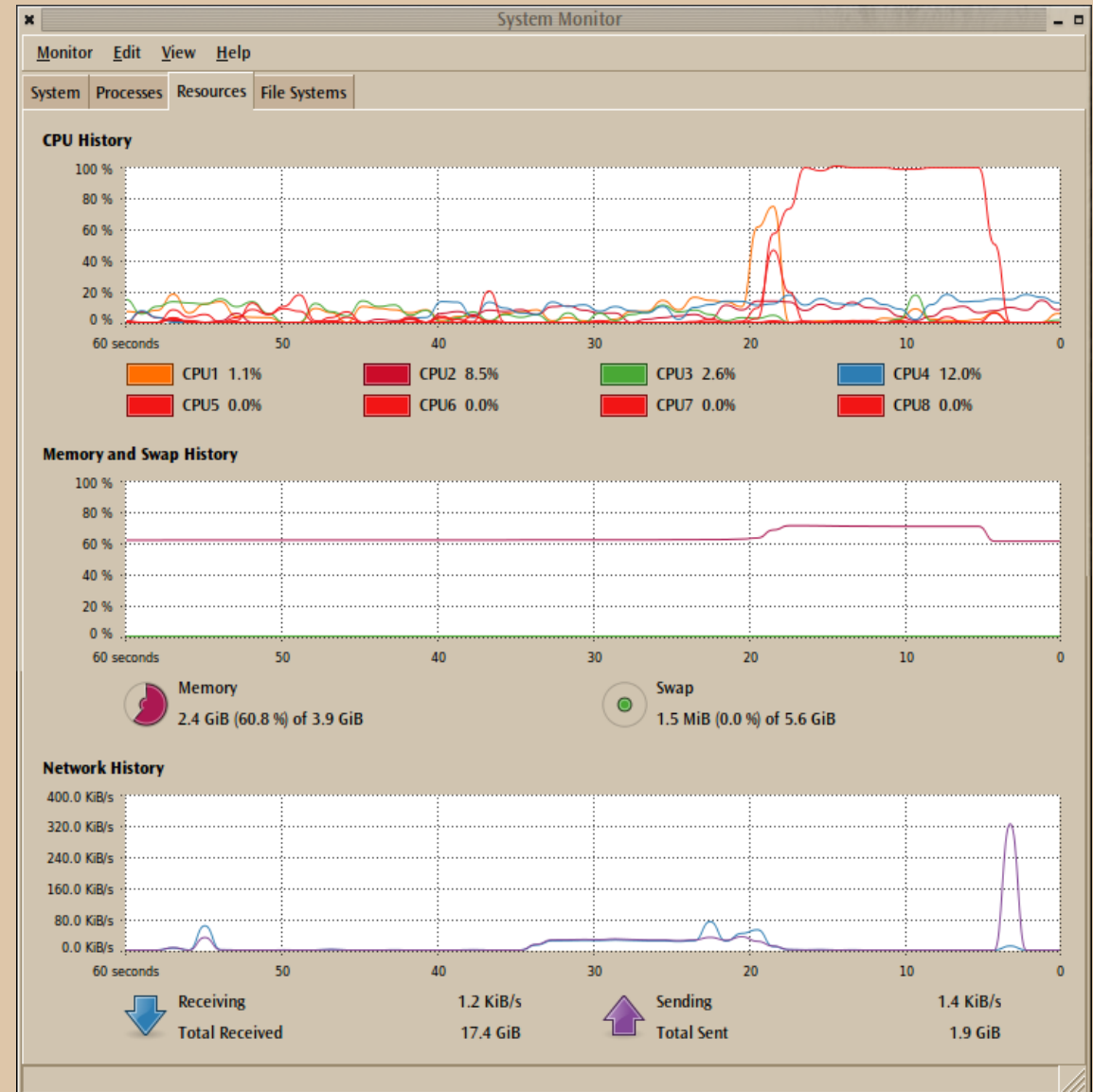




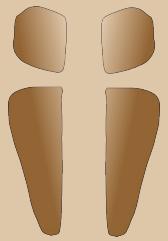
Groovy Sequential

Code

pi = 3.1415926535904264
iteration count = 100000000
elapse = 14.072310437



Must point out that there are only 10^8 not 10^9 iterations.



Groovy Futures

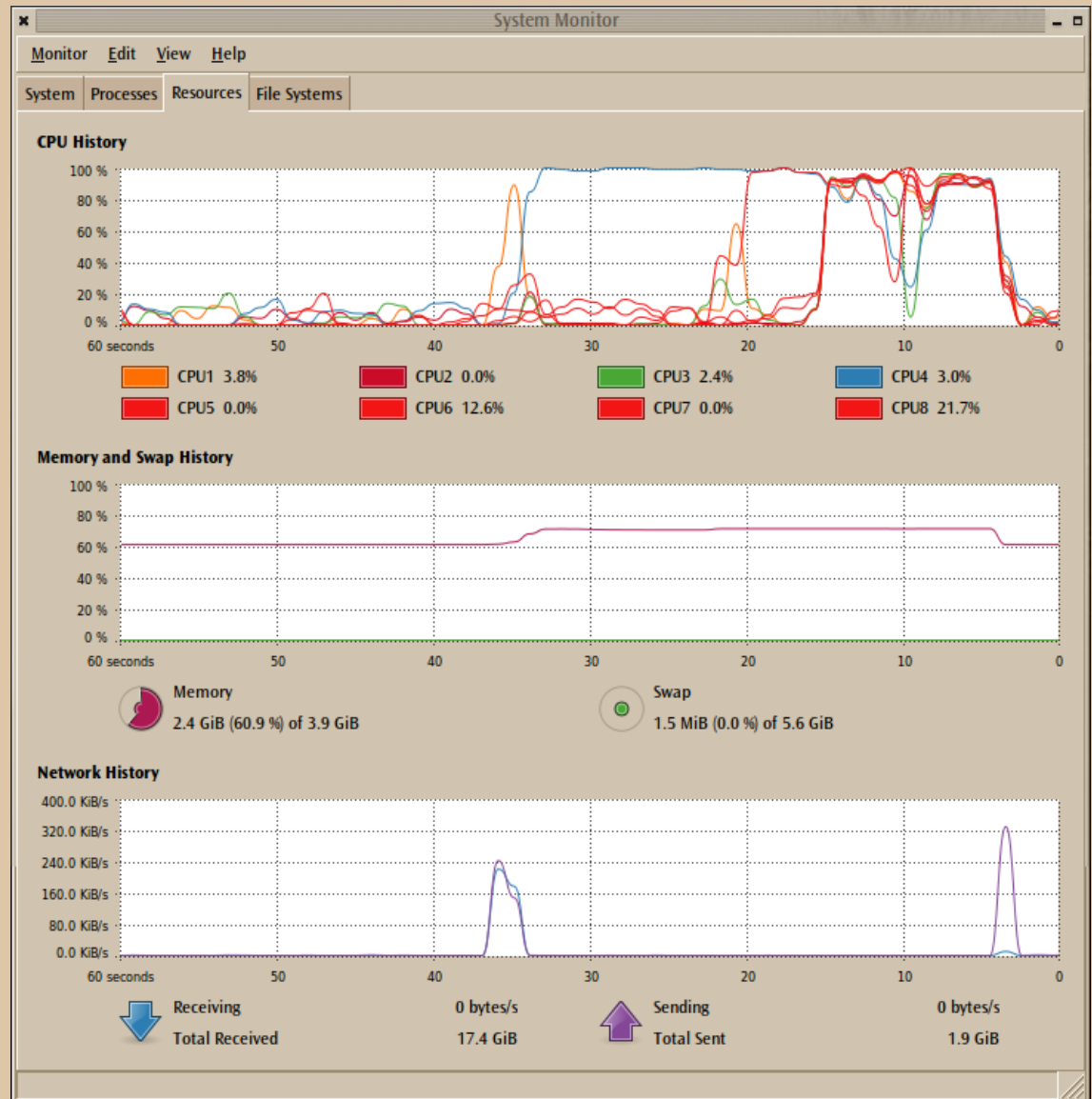
Code

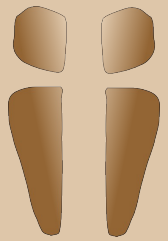
pi = 3.1415926535904264
iteration count = 100000000
elapsed = 12.276205552
processor count = 8
number of tasks = 1

pi = 3.14159265358991
iteration count = 100000000
elapsed = 6.425963471
processor count = 8
number of tasks = 2

pi = 3.141592653589815
iteration count = 100000000
elapsed = 6.233072074
processor count = 8
number of tasks = 8

pi = 3.1415926535898095
iteration count = 100000000
elapsed = 5.195073554
processor count = 8
number of tasks = 32





Groovy GPars Actors

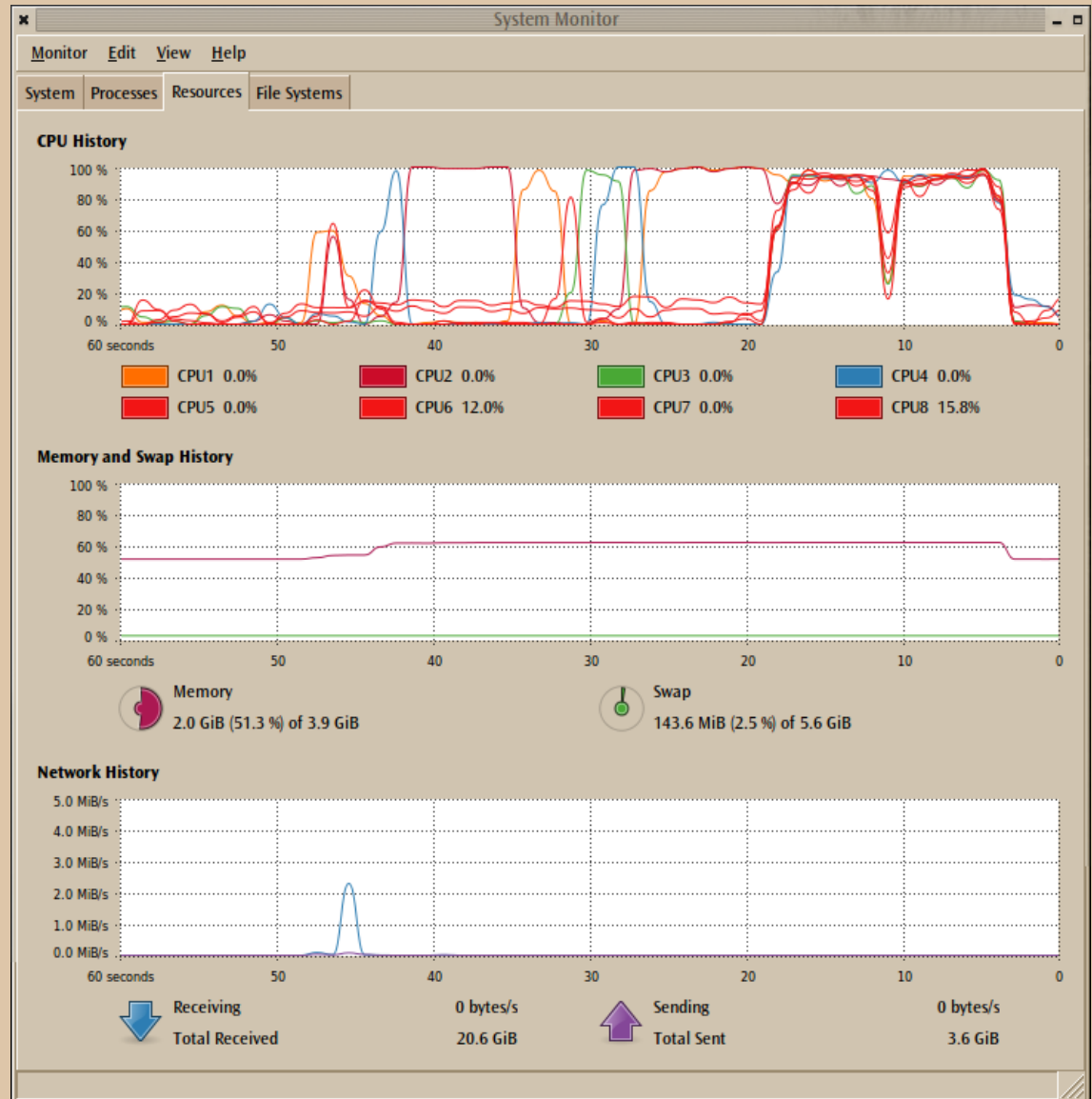
Code

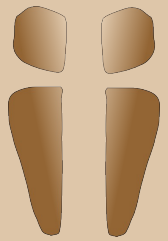
pi = 3.1415926535904264
 iteration count = 100000000
 elapse = 14.491816736
 processor count = 8
 actor count = 1

pi = 3.14159265358991
 iteration count = 100000000
 elapse = 11.198631097
 processor count = 8
 actor count = 2

pi = 3.1415926535898158
 iteration count = 100000000
 elapse = 7.621866979
 processor count = 8
 actor count = 8

pi = 3.141592653589808
 iteration count = 100000000
 elapse = 7.103628576
 processor count = 8
 actor count = 32





Groovy CSP (GPars)

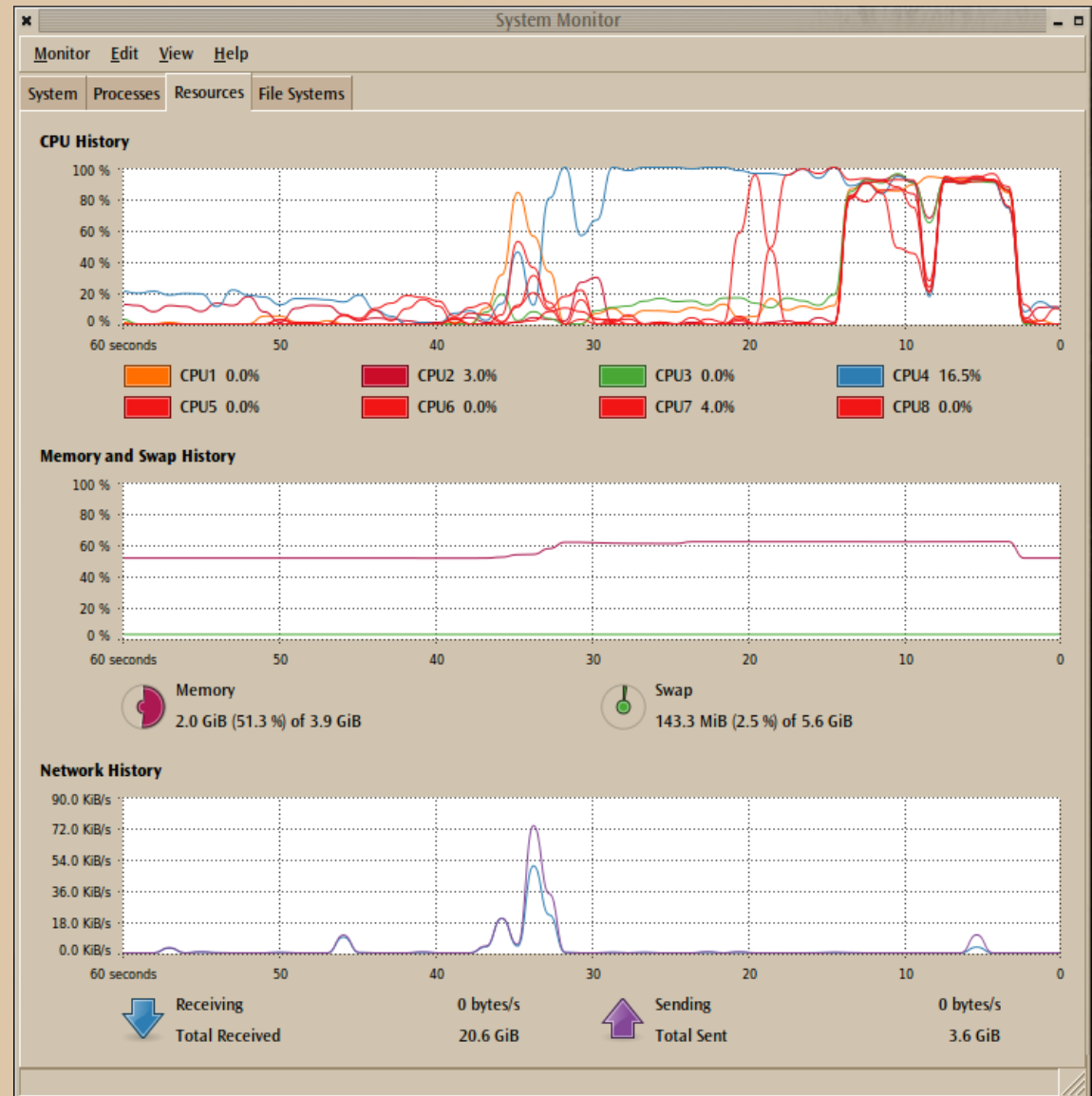
Code

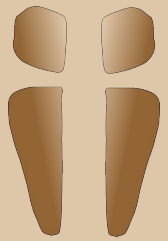
pi = 3.1415926535904264
 iteration count = 100000000
 elapse = 12.634194705
 processor count = 8
 task count = 1

pi = 3.14159265358991
 iteration count = 100000000
 elapse = 6.625233707
 processor count = 8
 task count = 2

pi = 3.141592653589815
 iteration count = 100000000
 elapse = 5.779894487
 processor count = 8
 task count = 8

pi = 3.1415926535898095
 iteration count = 100000000
 elapse = 5.36017772
 processor count = 8
 task count = 32





Groovy GPars GParsPool

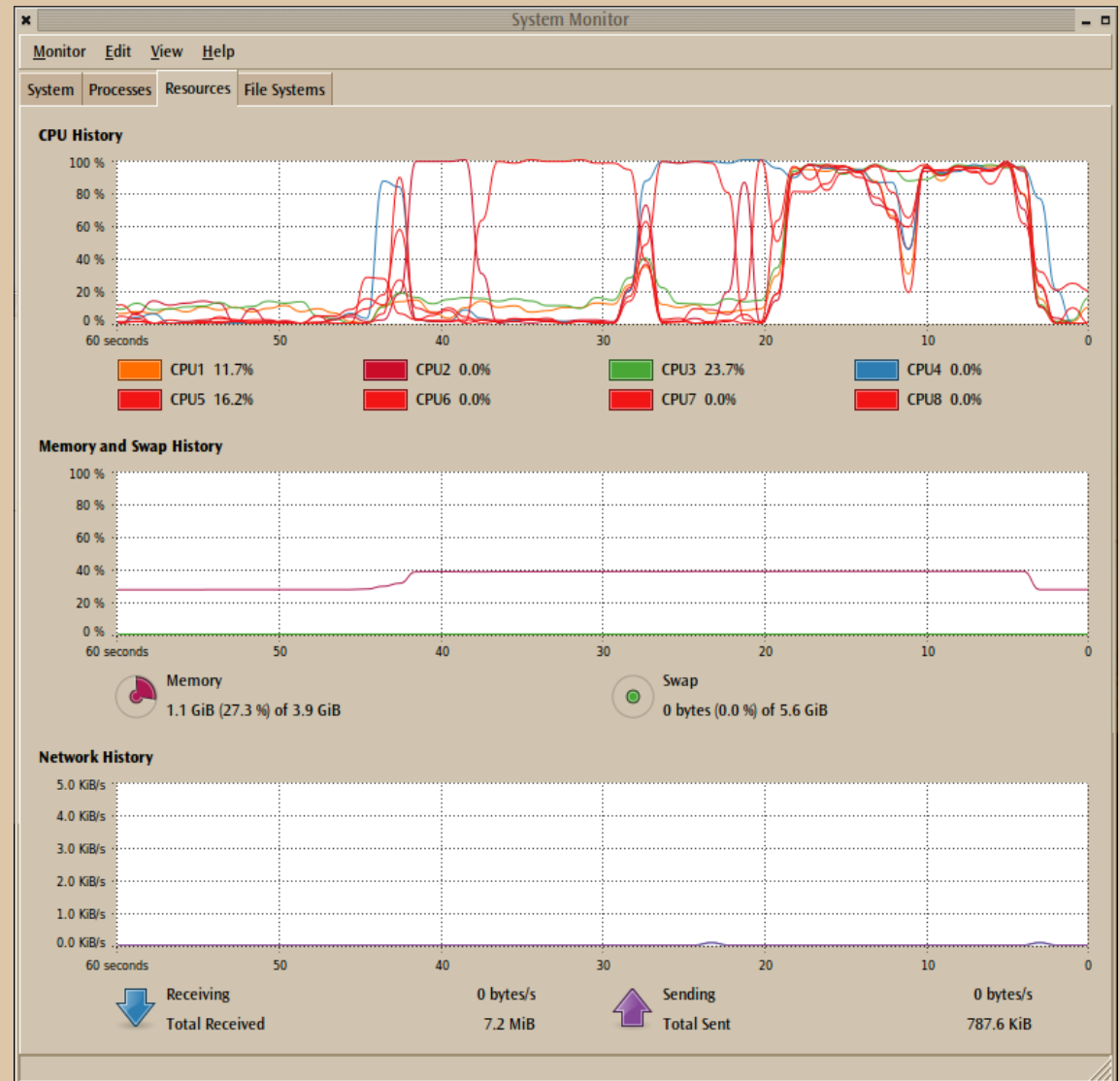
Code

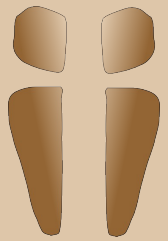
pi = 3.1415926535904264
 iteration count = 100000000
 elapse = 13.998522143
 processor count = 8
 task count = 1

pi = 3.14159265358991
 iteration count = 100000000
 elapse = 8.963442228
 processor count = 8
 task count = 2

pi = 3.141592653589815
 iteration count = 100000000
 elapse = 7.988819176
 processor count = 8
 task count = 8

pi = 3.1415926535898078
 iteration count = 100000000
 elapse = 7.811651308
 processor count = 8
 task count = 32





Groovy GPars Parallel Enhancer

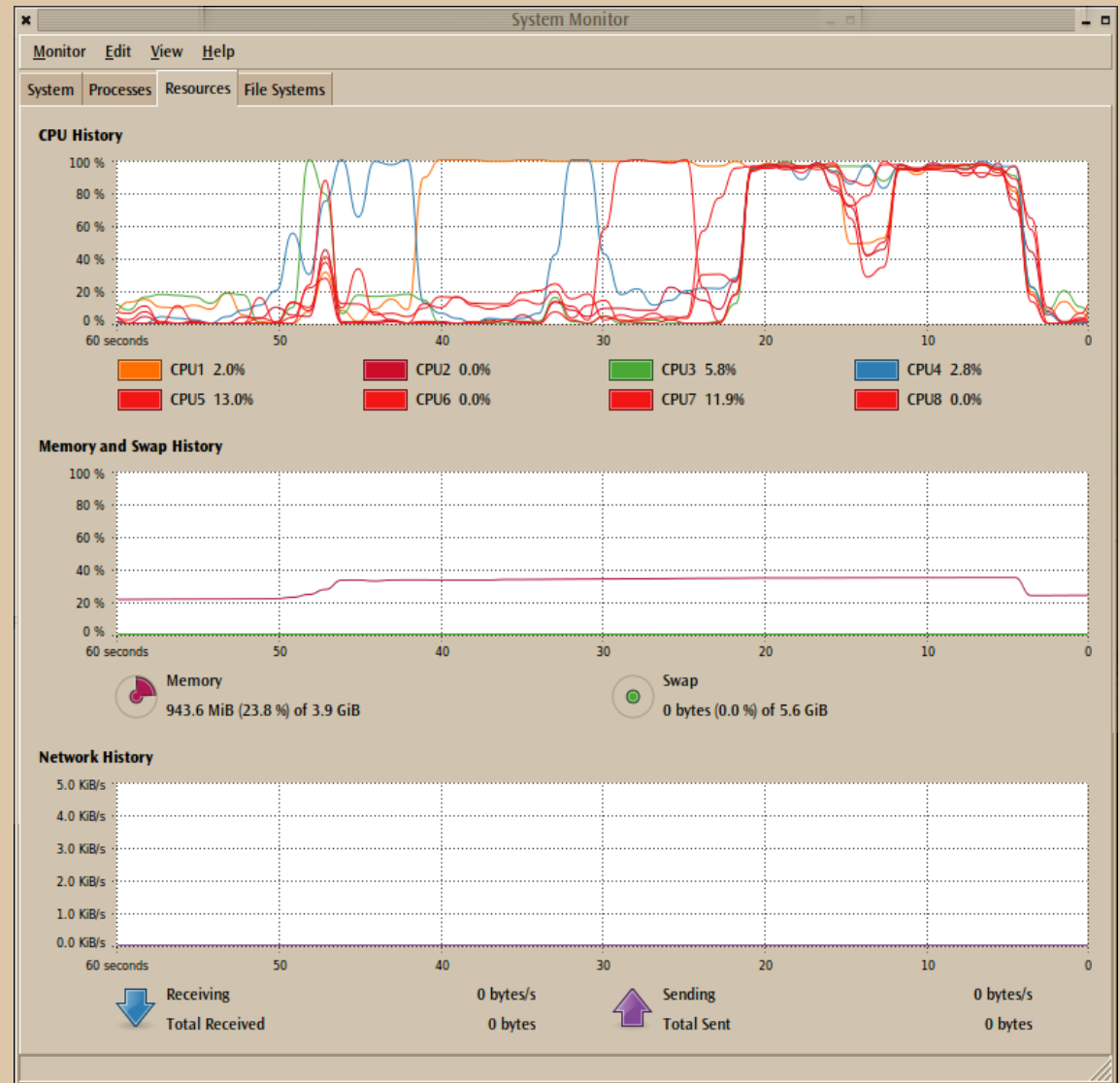
Code

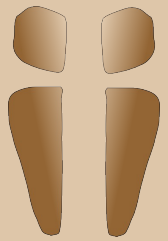
pi = 3.1415926535904264
 iteration count = 100000000
 elapse = 14.16521381
 processor count = 8
 task count = 1

pi = 3.14159265358991
 iteration count = 100000000
 elapse = 11.16217283
 processor count = 8
 task count = 2

pi = 3.141592653589815
 iteration count = 100000000
 elapse = 8.936376137
 processor count = 8
 task count = 8

pi = 3.1415926535898078
 iteration count = 100000000
 elapse = 8.77617131
 processor count = 8
 task count = 32





Groovy/Java CSP (GParS)

Groovy Code

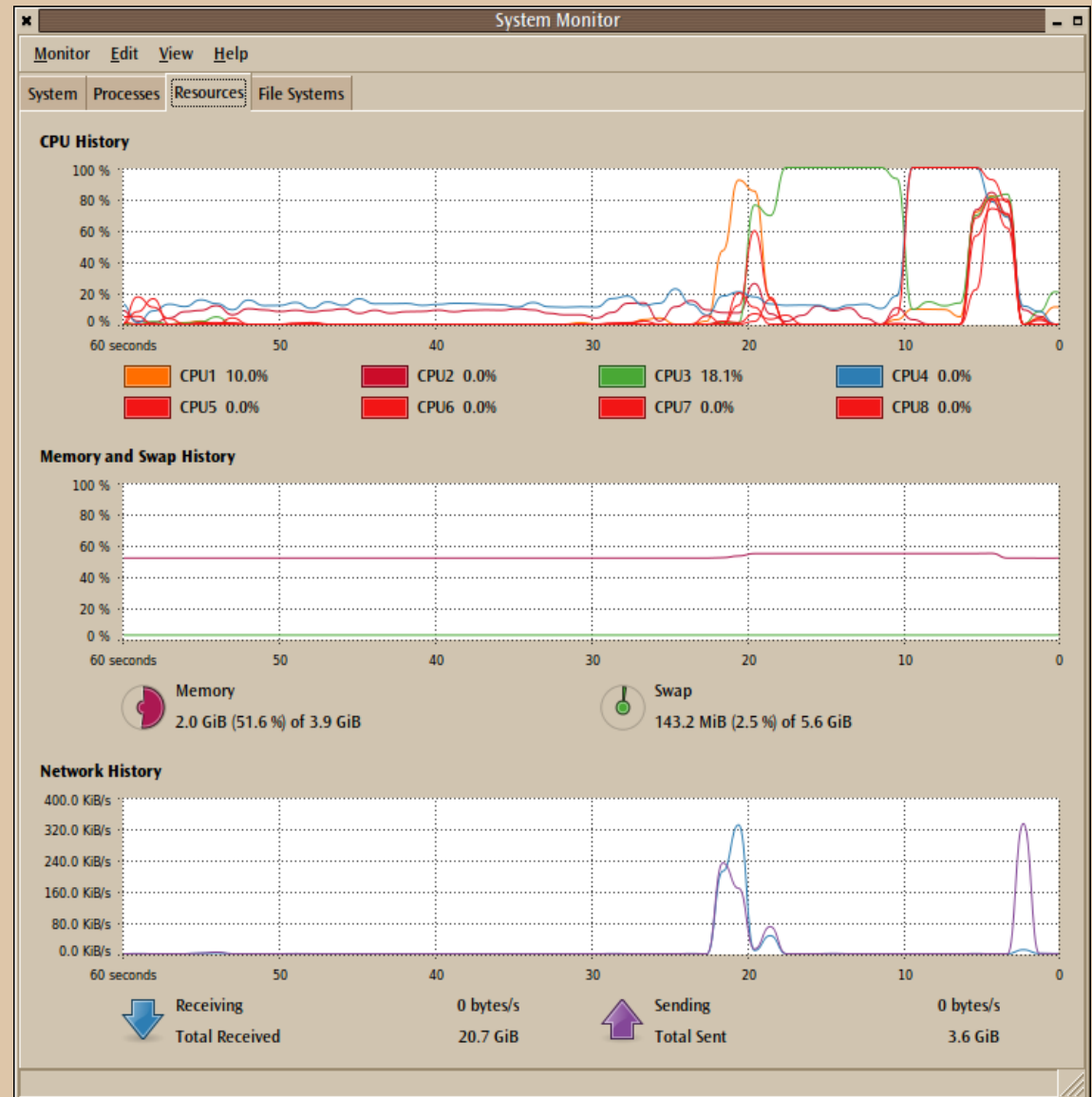
Java Code

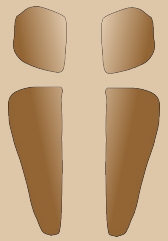
pi = 3.1415926535899708
 iteration count = 1000000000
 elapse = 8.824555439
 processor count = 8
 task count = 1

pi = 3.141592653589901
 iteration count = 1000000000
 elapse = 4.305456728
 processor count = 8
 task count = 2

pi = 3.1415926535897687
 iteration count = 1000000000
 elapse = 1.249605895
 processor count = 8
 task count = 8

pi = 3.141592653589758
 iteration count = 1000000000
 elapse = 1.276993013
 processor count = 8
 task count = 32





Groovy/Java GPars GParsPool

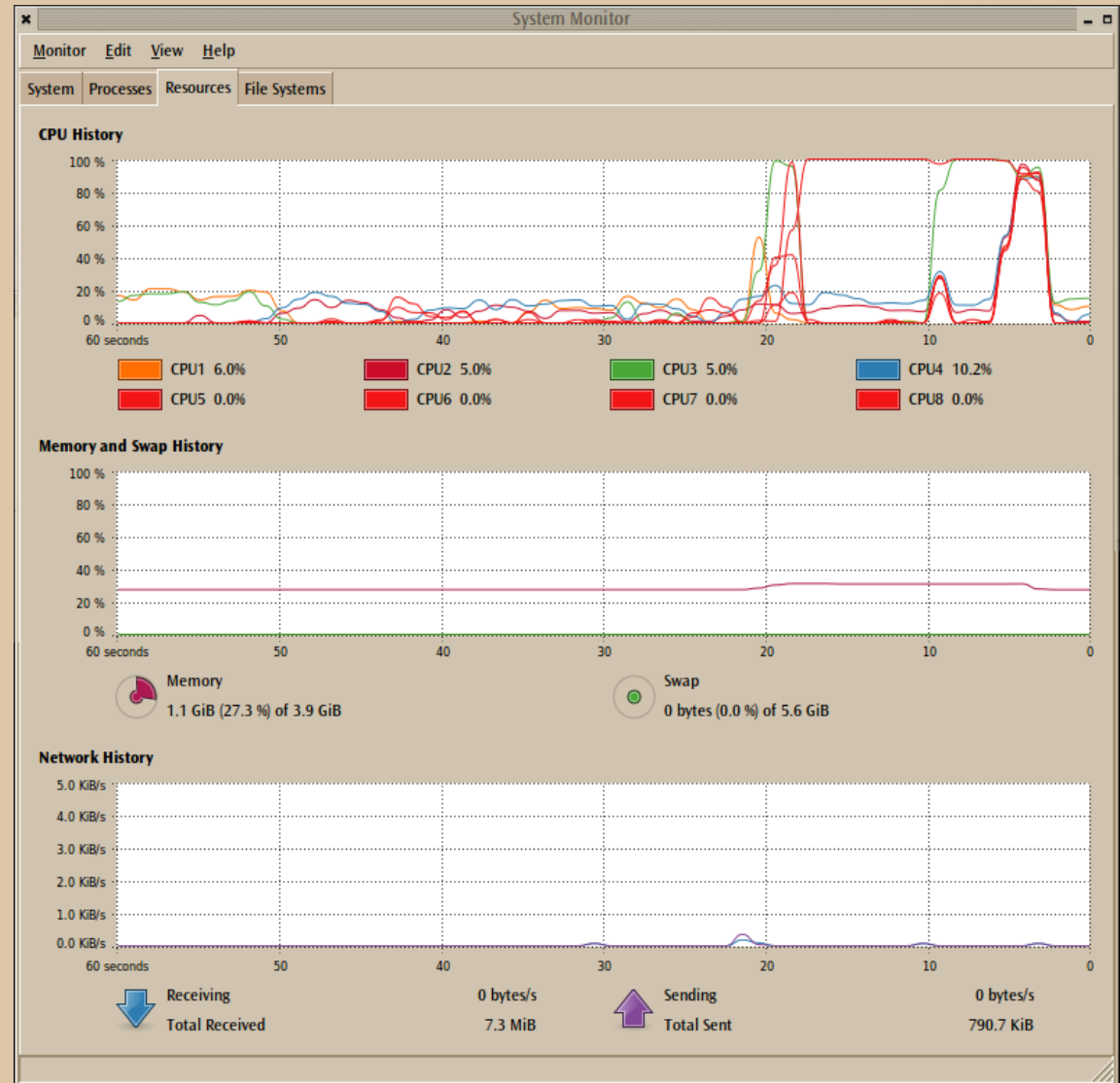
Code

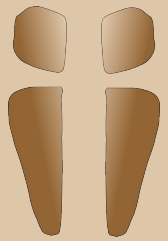
pi = 3.1415926535899708
 iteration count = 1000000000
 elapse = 8.793804027
 processor count = 8
 task count = 1

pi = 3.141592653589901
 iteration count = 1000000000
 elapse = 4.300636103
 processor count = 8
 task count = 2

pi = 3.1415926535897687
 iteration count = 1000000000
 elapse = 1.189305908
 processor count = 8
 task count = 8

pi = 3.141592653589757
 iteration count = 1000000000
 elapse = 1.235070087
 processor count = 8
 task count = 32





Groovy/Java GPars ParallelEnhancer

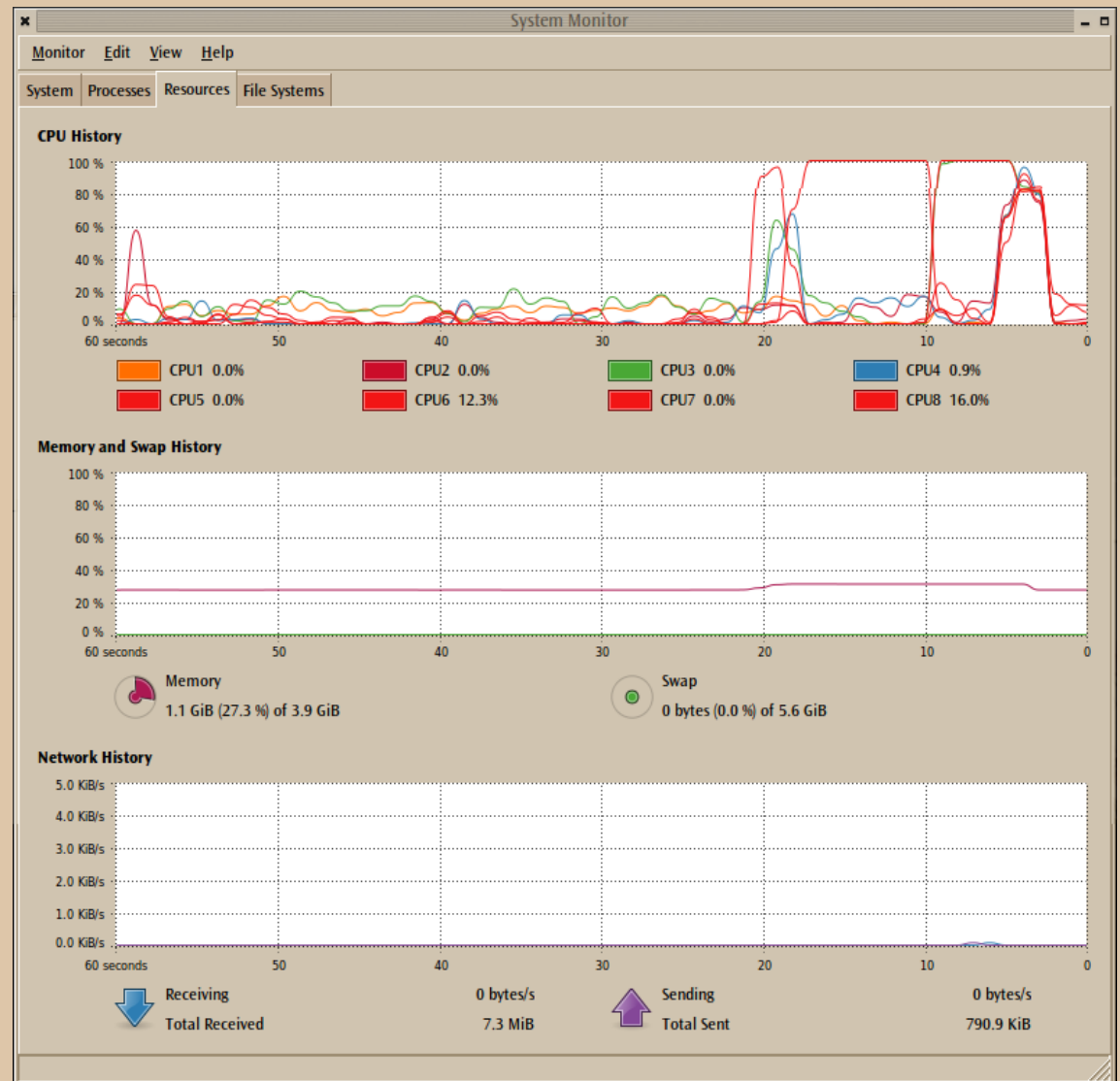
Code

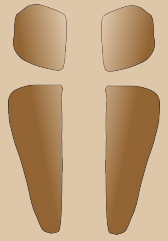
pi = 1.2870022175863478
 iteration count = 1000000000
 elapse = 8.842858931
 processor count = 8
 task count = 1

pi = 2.076584456986047
 iteration count = 1000000000
 elapse = 4.308636961
 processor count = 8
 task count = 2

pi = 2.8791959662656192
 iteration count = 1000000000
 elapse = 1.254583202
 processor count = 8
 task count = 8

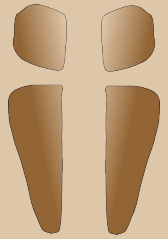
pi = 3.078166926991062
 iteration count = 1000000000
 elapse = 1.213908597
 processor count = 8
 task count = 32





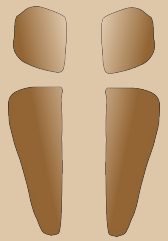
Other Languages

- Scala
- *X10*
- *Habanero-Java*
- *Fortress*
- Clojure



Summary

- The JVM has it all: commoditizes processor and operating system.
- Languages on the JVM are efficient enough for real computations.
- Parallelism is supported on the JVM.
- The JVM is the platform of choice.

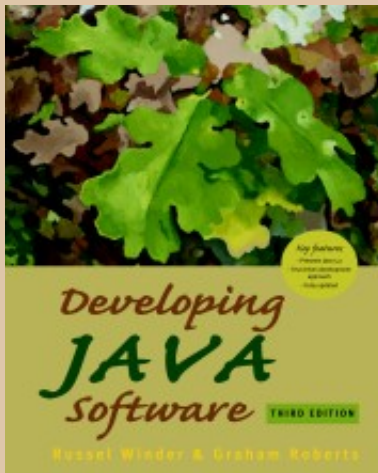
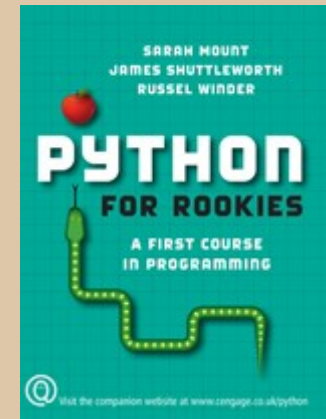


Blatant Advertising

Python for Rookies

Sarah Mount, James Shuttleworth and
Russel Winder

Thomson Learning *Now called Cengage Learning.*



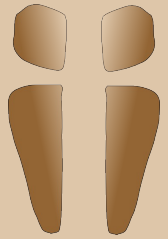
Developing Java Software Third Edition

Russel Winder and Graham Roberts

Wiley

Buy these books!





Gpars
Gpars

is where the
(parallel)
action is.

<http://gpars.codehaus.org>