PostgreSQL on VAX!

Or....
What I did for fun during my summer vacation!
VAX?
# MicroVAX 3800

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative Performance x VAX-11/780 (1 MIP)</td>
<td>3.8</td>
</tr>
<tr>
<td>Number of Processors</td>
<td>1</td>
</tr>
<tr>
<td>Max. Memory Support</td>
<td>64 MB</td>
</tr>
<tr>
<td>Max. Local Disk Capacity (formatted)</td>
<td>MV 3800: 2.4 GB; MV 3900: 9.7 GB</td>
</tr>
<tr>
<td>Max I/O Throughput</td>
<td>3.3 MB/s</td>
</tr>
<tr>
<td>Floating Point Accelerator</td>
<td>Standard</td>
</tr>
<tr>
<td>Floating Point Data Types</td>
<td>F, D, G, H</td>
</tr>
<tr>
<td>Cache Size</td>
<td>1 KB on chip; 64 KB on board</td>
</tr>
</tbody>
</table>
Where to find a VAX?

DEC - VAX 3800/3900 CPU BD/KA655-AA
by DEC
Be the first to review this item

Currently unavailable.
We don't know when or if this item will be back in stock.
simh - VAX emulator

The Computer History Simulation Project
http://simh.trailing-edge.com/

Installing NetBSD on a ka655x VAX 3800:
http://www.netbsd.org/ports/vax/emulator-howto.html
Operating systems that support VAX

- PDP-11 UNIX V5
- PDP-11 UNIX V6
- PDP-11 UNIX V7
- VAX/VMS
- VAX/Rdb

Operating systems PostgreSQL supports

- FreeBSD
- Windows
- Solaris
- Mac OS
- AIX
- NetBSD
- OpenBSD
Install NetBSD

Time passes….

24 hours later…
Install pkgsrc

Time passes....

Run out of disk space... drives are limited to 2.4GB ... create new drive
Time passes....

Ran out of inodes ... create new file system
Time passes...

48 hours later....
Time passes…
Ran out of space again… create 20G NFS volume from host machine and mount it from guest machine …
Time passes…

72 hours later…
Build Postgres!

Time passes....

48 hours later
Run regression tests...

Kernel panic (probably a NetBSD bug that still warrants some investigation):

panic: usrptmap space leakage
cpu0: Begin traceback...
panic: usrptmap space leakage
Stack traceback:
  Process is executing in user space.
cpu0: End traceback...

Out of memory ... initdb's smallest numbers are still too large ...
Reduce max_backends and run tests with MAX_CONNECTIONS=2

It took 7h20m to run the regression tests
No IEEE Floating Point

Expected but the consequences are surprising.

On a modern architecture with IEEE floating point:

$ gcc -Wall exp.c -lm
$ ./a.out
exp(88.0297) = 1.70141e+38

On VAX:

simh$ gcc -Wall exp.c -lm
simh$ ./a.out
[4] Illegal instruction (core dumped) ./a.out

Oh well, Postgres documents that users should expect the floating point semantics of the architecture, so job done? No, Postgres doesn't catch SIGILL or override infnan() so it dies rather than raise a floating point error.
Infinite loop in GROUPING SETS test

Regression:

```
commit 44ed65a545970829322098e22d10947e6d545d9a
Author: Tom Lane <tgl@sss.pgh.pa.us>
Date:   Sun Aug 23 13:02:13 2015 -0400

Avoid use of float arithmetic in bipartite_match.c.

Since the distances used in this algorithm are small integers (not more than the size of the U set, in fact), there is no good reason to use float arithmetic for them. Use short ints instead: they're smaller, faster, and require no special portability assumptions.

Per testing by Greg Stark, which disclosed that the code got into an infinite loop on VAX for lack of IEEE-style float infinities. We don't really care all that much whether Postgres can run on a VAX anymore, but there seems sufficient reason to change this code anyway.
```
Reduce number of bytes examined by convert_one_string_to_scalar().

Previously, convert_one_string_to_scalar() would examine up to 20 bytes of
the input string, producing a scalar conversion with theoretical precision
far greater than is of any possible use considering the other limitations on
the accuracy of the resulting selectivity estimate. (I think this choice
might pre-date the caller-level logic that strips any common prefix of the
strings; before that, there could have been value in scanning the strings far
enough to use all the precision available in a double.)

Aside from wasting cycles to little purpose, this choice meant that the
"denom" variable could grow to as much as 256^21 = 3.74e50, which could
overflow in some non-IEEE float arithmetics. While we don't really support
any machines with non-IEEE arithmetic anymore, this still seems like quite an
unnecessary platform dependency. Limit the scan to 12 bytes instead, thus
limiting "denom" to 256^13 = 2.03e31, a value more likely to be computable
everywhere.

Per testing by Greg Stark, which showed overflow failures in our standard
regression tests on VAX.
Goal was to add new build farm member building Postgres regularly and testing it on VAX architecture (even if emulated).

But that’s hopeless. We would never pass the regression tests without significantly weakening our testing.

Without a build farm member we can’t seriously say we “support” VAX :(