CSE 380: HW 0

Evan Ott
UT EID: eao466

September 30, 2016

0.0

See Figures 1 and 2 along with Tables 1 and 2.

The first set of data could indicate plausibly one of two things: Italy and Saudi Arabia are competitive and want to have the best supercomputers, or that there are (or were) features at the time in the country that make it easy to build the best supercomputer (cheap electricity, increased funding, etc.).

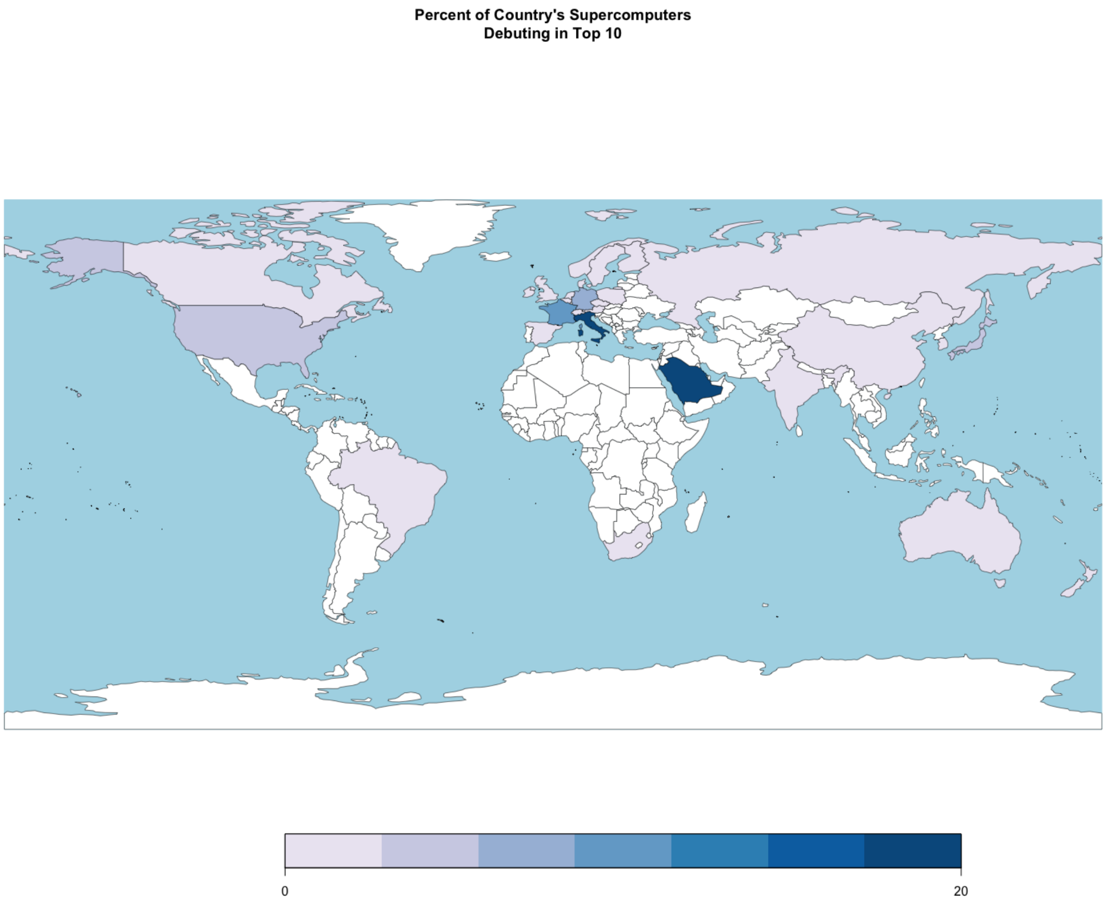

Figure 1: Percent of each country's supercomputers that debuted in the top 10.

The second set of data shows that in terms of applications, researchers and vendors may have a little bit more pride in their supercomputer rankings than those in academia or general industry. Vendors have a major incentive to have the best supercomputer ("look, our chips are the best!"), whereas academia doesn't (they just need it to work, and will take what funding they can get from NSF to get things done). An interesting note here is that there is only one on the list that is classified, and it debuted at rank 150. As for narrative with that one, it would seem that governments doing classified work keep their supercomputers' specifications secret as to not reveal their actual capabilities. So that's fun.

Country	Value
Italy	20.00
Saudi Arabia	20.00
France	11.11
Germany	7.69
United States	5.45
Japan	3.45
China	2.38

Table 1: Percent of each country's supercomputers that debuted in the top 10. Countries on the top 500 that have not debuted in the top 10: Australia, Austria, Belgium, Brazil, Canada, Czech Republic, Denmark, Finland, India, Ireland, Korea, South, Netherlands, New Zealand, Norway, Poland, Russia, Singapore, South Africa, Spain, Sweden, Switzerland, United Kingdom.

Segment	Value
Research	25.23
Vendor	14.29
Government	11.63
Academic	9.57
Industry	1.23
Classified	0.00

Table 2: Percent of segment's supercomputers debuting in top 20.

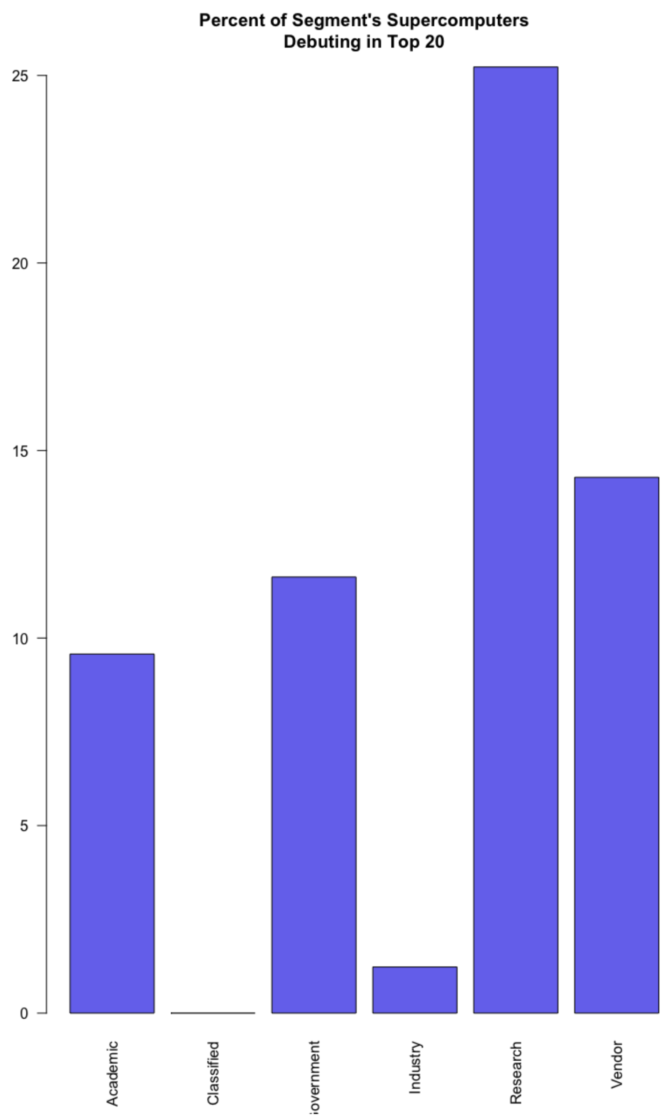

Figure 2: Percent of segment's supercomputers debuting in top 20.

```r
library(rworldmap)
library(reshape2)
library(xtable)

top500 = read.csv("/Users/Evan/Dropbox/A-Grad-School/CSE380/TOP500_201606.csv")
countryData = 100 * colSums(xtabs(~ top500$First.Rank + top500$Country)[1:9,]) /
  colSums(xtabs(~ top500$First.Rank + top500$Country))

countryData = melt(countryData)

df = data.frame(Country = labels(countryData)[[1]],
           Value = countryData[[1]])

#join to a coarse resolution map
spdf <- joinCountryData2Map(df,
                            joinCode = "NAME",
                            nameJoinColumn = "Country")

mapCountryData(spdf,
               nameColumnToPlot = "Value",
               catMethod = "fixedWidth",
               oceanCol = 'lightblue',
               colourPalette = brewer.pal(9, "PuBu")[2:8],
               missingCountryCol = '#ffffff',
               borderCol = 'black',
               mapTitle = 'Percent of Country\'s Supercomputers\nDebuting in Top 10')

print(xtable(subset(df[order(df$Value, decreasing = TRUE),], Value > 0)), include.rownames = FALSE)




segmentData = 100 * colSums(xtabs(~ top500$First.Rank + top500$Segment)[1:19,]) /
  colSums(xtabs(~ top500$First.Rank + top500$Segment))

segmentData = melt(segmentData)

segmentdf = data.frame(Segment = labels(segmentData)[[1]],
                Value = segmentData[[1]])
barplot(segmentdf$Value,
        names.arg = segmentdf$Segment,
        col = '#7777ee',
        main = "Percent of Segment\'s Supercomputers\nDebuting in Top 20", las=2)

print(xtable(segmentdf[order(segmentdf$Value, decreasing = TRUE), ]), include.rownames = FALSE)
```

0.1

A Stampede core is "faster" than a Lonestar 4 core because it can do twice as many floating-point operations per clock period. To show a full comparison:

System	Clock Speed (GHz)	flops / cp	Gflops / second (1 core)	cores / node	Gflops / node (all cores)
Lonestar 4	**3.33**	4	13.32	12	159.84
Lonestar 5	2.6	**8**	20.8	**24**	**499.2**
Stampede	2.7	**8**	**21.6**	16	345.6

So in terms of floating-point operations per second (per core), Stampede is faster than Lonestar 5 is faster than Lonestar 4. However, in terms of peak performance per node, Lonestar 5 actually wins, followed by Stampede, then by Lonestar 4. All this even though Lonestar 4 has the fastest clock speed.

0.2

- How many operations per clock cycle?
  - Intel Xeon E5-2680: 8 Flop / cp
- Clock rate?
  - 2.7 GHz
- 8 Flop / cp * 2.7 GHz = 21.6 GFlops
- 8-way set associative L1 cache [1]. Assume 2 W / cp.
  - 2 * 2.7 * 8 = 43.2 GB/s
- Assume $\sim$ 0.25 W/cp from main memory
  - 0.25 * 2.7 * 8 = 5.4 GB/s

0.3

```
# getent passwd | cut -f 7 -d: | sort | uniq
```

The above command works as follows:

- bash reads the command, finding no splats or parts of the command that need to be filled in.
- `getent` is called to query the Name Service Switch libraries to get the passwd database
- `getent` returns (to standard output) a line for each user including username, group, name, home directory and shell, separated by colons.
- The pipe redirects this standard output to standard input for `cut`, which takes the seventh field as separated by the ":" delimiter, which in the case of `getent passwd` is the location of the shell, and prints it to standard output.
- The next pipe redirects to `sort`, which sorts the list alphanumerically. It outputs the same number of lines as the input, but sorted.
- The final pipe redirects to `uniq` which merges adjacent matching lines. As such, since the lines are already sorted and all the matching lines will be next to each other, `uniq` will output each shell used on stampede exactly once.

0.4

```
# getent passwd | cut -f 7 -d: | sort | uniq --count | sort -bgr
  11504 /bin_bash
   8949 /bin/bash
    136 /bin_tcsh
     80 /bin/tcsh
     31 /sbin/nologin
     22 /bin/csh
     20 /bin_csh
     18 /bin/zsh
     13 /bin_zsh
      1 /sbin/shutdown
      1 /sbin/halt
      1 /bin/sync
```

So 8948 active users on Stampede use bash, 80 use tcsh and the next most popular actual shell is csh.

0.5

```
# getent passwd | cut -f 1 -d: | awk '{print length(), $0}' | sort -g | tail -n1
13 avahi-autoipd
```

avahi-autoipd is the longest TACC username, at 13 characters. It's the Avahi IPv4LL Stack (from `finger avahi-autoipd`), so it's a user, just not necessarily a person.

```
# getent passwd | cut -f 1 -d: | awk '{print length()}' | grep "8" | wc -l
10051
```

10051 users have an 8-character username.

0.6

Question: How many usernames are a person's full name (first and last words in their name)?

```
# getent passwd | cut -f 1,5 -d: --output-delimiter " " | awk '{split($0, a, " "); \
username=a[1]; name = substr(tolower(sprintf("%s%s", a[2], a[length(a)])), 1, 8);\
f1 = index(username, name); f2 = index(name, username); print((f1 == f2) && (f1 == 1))}'\
| grep 1 | wc -l
766
```

0.7

0.7.1

```
# cat words | grep "^[a-zA-Z]" | awk '{print tolower(substr($0, 0, 1))}' | \
uniq --count | sed 's/ //g' | \
awk '{print substr($0, length($0), length($0)) ":" substr($0, 0, length($0) - 1)}'
a:31788
b:25192
c:40042
d:23263
e:17149
f:16263
g:15075
h:19317
i:15377
j:4426
k:6490
l:14680
m:26175
n:16668
o:15274
p:42073
q:3274
r:21688
s:51774
t:25875
u:23930
v:7019
w:12023
x:631
y:1820
z:1914
```

0.7.2

```
# cat words | awk '{print tolower($0)}' | grep "s$" | wc -l
```

```
86041
# cat words | awk '{print tolower($0)}' | grep "'s$" | wc -l
0
```

The dictionary has 86,041 words that end in "s" but zero that are possessive (ends in "'s")

0.7.3

```
# cat words | awk '{print tolower($0)}' | grep "foo[^dlt]\|foo$" | wc -l
37
```

There are 37 words that contain the substring "foo" but not "food", "fool" or "foot". In other words, the words contain "foo" followed by any character other than "d", "l", or "t" or "foo" must be the end of the string.

0.7.4

Question: How many prefixes are there in the dictionary? Suffixes? What words have at least 4 dashes in them that aren't contiguous that aren't suffixes nor prefixes (i.e., cannot start nor end with a dash and have non-dashes in-between all dashes)?

```
# cat words | grep "\-$" | wc -l
1824
# cat words | grep "^-" | wc -l
560
# cat words | grep "[^-]\{1,\}-[^-]\{1,\}-[^-]\{1,\}-[^-]\{1,\}-[^-]\{1,\}"
Jack-go-to-bed-at-noon
knock-down-and-drag-out
Mentor-on-the-Lake-Village
what-do-you-call-it
what-you-may-call-'em
```

Related, looking at the 3-dash version of the above yields some great words such as "king-of-the-salmon" (a particular fish), "stick-to-it-iveness" (didn't know it was a proper word), "toad-in-the-hole" (British dish) and "wag-on-the-wall" (wall clock with elements exposed). I dunno, I thought these were esoteric enough to bring up as things more people should know.

I can't stop myself. What's the longest prefix? Longest suffix?

```
# cat words | grep "\-$" | awk '{print length, $0}' | sort -n| tail -n1
13 philosophico-
# cat words | grep "^-" | awk '{print length, $0}' | sort -n| tail -n1
10 -morphosis
```

0.8

There is storage on each login node (`/tmp`), but ideally most or all data and work should be done in one of the following three filesystems. `/tmp` is really for local I/O only, and usually `$SCRATCH` should be used in most cases instead.

`$HOME` is permanent, has a quota and is backed up. It's used for storing code, executibles, things of that nature.

`$WORK` is permanent, has a quota, but not backed-up. Code can be run here, and large files can be stored here.

`$SCRATCH` is high-speed purged storage. It's a temporary workspace. Code can be run here, and large files can be stored here.

All three are Lustre parallel filesystems on 72 Dell R610 data servers through InfiniBand with four Dell R710 meta data servers and two Dell MD 3220 Storage Arrays. As stated on the TACC website, "the interconnect is an FDR InfiniBand network of Mellanox switches, consisting of a fat tree topology of eight core-switches and over 320 leaf switches with a 5/4 oversubscription."

There's also `$ARCHIVE` which is tape storage through Ranch.

0.9.1

```
# uname -r
2.6.32-431.17.1.el6.x86_64
```

The version is 2.6.32

0.9.2

```
# curl https://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.32.27.tar.gz > \
linux-2.6.32.27.tar.gz && gunzip linux-2.6.32.27.tar.gz && tar -xvf linux-2.6.32.27.tar
```

(I like seeing all the files as it unarchives them).

0.9.3

```
# du -hc | tail -n1
418M total
# find . | wc -l
   32365
# find . | grep '\.c$' | wc -l
   13149
```

This version of Linux is 418 megabytes. Including directories (everything is a file or process!), this version has 32365 files, 13149 of which are .c files.

0.9.4

```
# ls -R | grep "^\..*:$" | wc -l
1878
# find . -type d | wc -l
1878
# tree -d -i --noreport | wc -l
1878
```

0.9.5

```
# find . | grep '\.[hc]$' | xargs wc -l | sort -bg | grep "\.[hc]$" | tail -n1
   17722 ./sound/pci/hda/patch_realtek.c
```

The /sound/pci/hda/patch_realtek.c file is the largest (in terms of lines of code) and has 17,722 lines.

0.9.6

```
# find . | grep '\.[hc]$' | xargs wc -l | tail -n1
 828952 total
```

Including comments, there are 828,952 lines of code in this version of the Linux kernel.

0.9.7

Question: How many `main` functions are there? How many versions have the canonical argument names `int argc` and `char *argv`?

```
# find . | grep '\.[hc]$' | xargs grep "int main(int .*, char .*)"
   51
# find . | grep '\.[hc]$' | xargs grep "int main(int argc, char \*argv\[\])" | wc -l
   27
```

References

[1] http://www.cpu-world.com/CPUs/Xeon/Intel-Xeon%20E5-2680.html