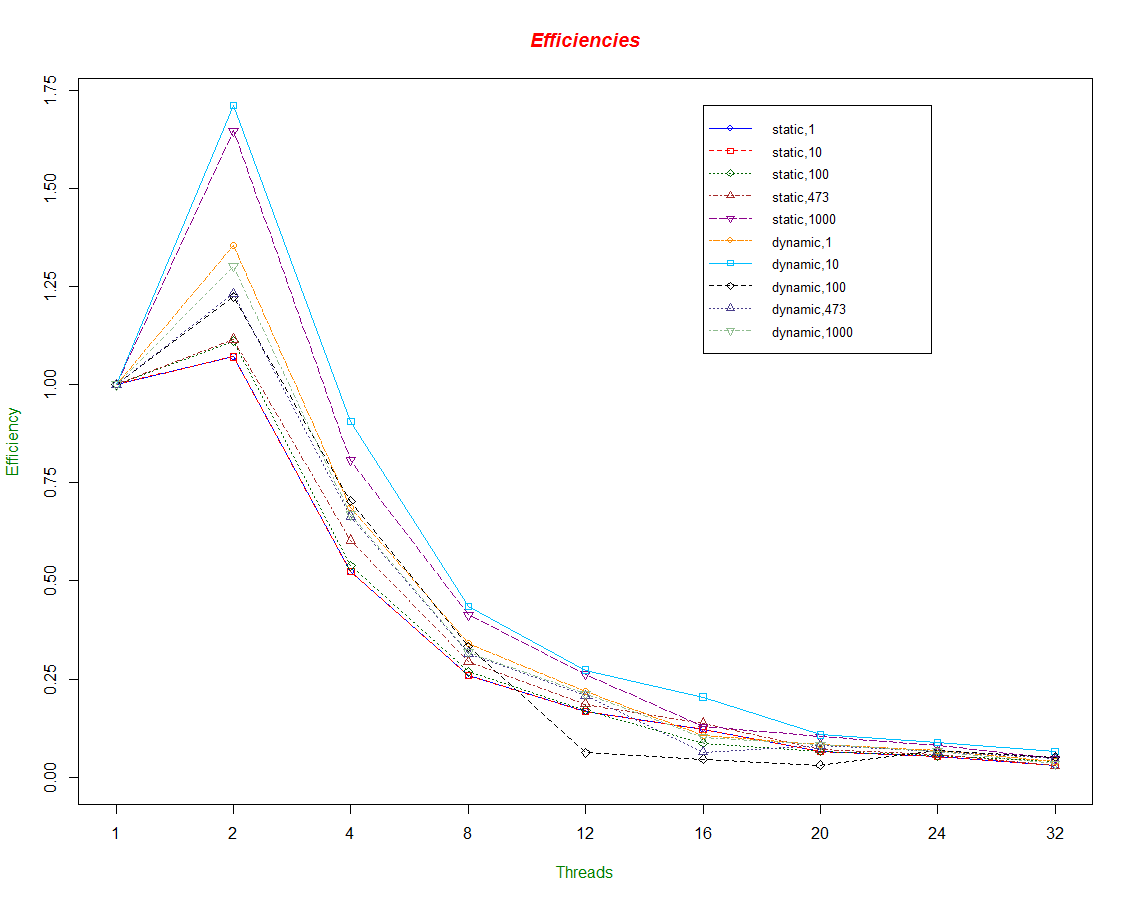
**Assignment 2 Writeup**

It seems to me that each method peaked at 2 threads, and then there was a drastic loss in efficiency across the board.I found that dynamic,10 worked best, closely followed by static,1000. I think it has to either do with Thor, or that the setup time go in the way when adding more and more threads.

I modeled this using R. I entered the datapoints into the program with the commands “rawX <-scan()” where X is the number I associated to the individual schedules. I then calculated the speedup for each with “sX<-TX/rawX” where X is still the number associated with the schedules, and TX is the time of one thread for the corresponding X. After that, I entered the number of threads into an array I called “p”. This allowed me to find the efficiency with “eX<-sX/p”. eX is then a single array such that eX[i]=sX[i]/p[i].

I’m only able to do this because I’m currently taking Stats using R, and this seemed nice to graph. The following lines contain the code to make the graph.

yRange<-range(0,e1,e2,e3,e4,e5,e6,e7,e8,e9,e10)

colors<-c("blue","red","darkgreen","brown","darkmagenta","darkorange","deepskyblue","gray0","darkslateblue","darkseagreen")

plot(e1, type="o", col=colors[1], ylim=yRange, axes=FALSE, ann=FALSE)

axis(1, at=1:9, lab=c("1","2","4","8","12","16","20","24","32"))

axis(2, at=c("0","0.25","0.5","0.75","1","1.25","1.5","1.75"))

box()

lines(e2, type="o", pch=22, lty=2, col=colors[2])

lines(e3, type="o", pch=23, lty=3, col=colors[3])

lines(e4, type="o", pch=24, lty=4, col=colors[4])

lines(e5, type="o", pch=25, lty=5, col=colors[5])

lines(e6, type="o", pch=21, lty=6, col=colors[6])

lines(e7, type="o", pch=22, lty=7, col=colors[7])

lines(e8, type="o", pch=23, lty=8, col=colors[8])

lines(e9, type="o", pch=24, lty=9, col=colors[9])

lines(e10, type="o", pch=25, lty=10, col=colors[10])

title(main="Efficiencies", col.main="red", font.main=4)

title(xlab="Threads", col.lab=rgb(0,0.5,0))

title(ylab="Efficiency", col.lab=rgb(0,0.5,0))

legend(6, yRange[2],c("static,1","static,10","static,100","static,473","static,1000","dynamic,1","dynamic,10","dynamic,100","dynamic,473","dynamic,1000"), cex=0.8,col=colors,pch=c(21,22,23,24,25,21,22,23,24,25),lty=c(1,2,3,4,5,6,7,8,9,10))

Datapoints:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Schedule\Threads | 1 | 2 | 4 | 8 | 12 | 16 | 20 | 24 | 32 |
| raw1 | static,1 | 0.841537 | 0.322771 | 0.296315 | 0.306048 | 0.32049 | 0.319292 | 0.516042 | 0.501791 | 0.741535 |
| raw2 | static,10 | 0.643157 | 0.300271 | 0.30634 | 0.310078 | 0.31911 | 0.331188 | 0.492555 | 0.507249 | 0.648927 |
| raw3 | static,100 | 0.662989 | 0.298567 | 0.307688 | 0.308837 | 0.32372 | 0.47797 | 0.504632 | 0.507204 | 0.512956 |
| raw4 | static,473 | 0.711946 | 0.319091 | 0.294737 | 0.302442 | 0.318474 | 0.324273 | 0.50711 | 0.51265 | 0.753785 |
| raw5 | static,1000 | 0.996751 | 0.302644 | 0.308619 | 0.3014 | 0.316726 | 0.486746 | 0.485855 | 0.506487 | 0.669955 |
| raw6 | dynamic,1 | 1.06995 | 0.395079 | 0.389786 | 0.392098 | 0.408802 | 0.630484 | 0.641215 | 0.646463 | 0.855276 |
| raw7 | dynamic,10 | 1.12557 | 0.328863 | 0.310679 | 0.323524 | 0.343737 | 0.344323 | 0.521019 | 0.531546 | 0.538283 |
| raw8 | dynamic,100 | 0.821185 | 0.335484 | 0.291982 | 0.308446 | 1.10317 | 1.108571 | 1.35878 | 0.509614 | 0.509259 |
| raw9 | dynamic,473 | 0.786673 | 0.319394 | 0.295748 | 0.311524 | 0.31557 | 0.769419 | 0.487078 | 0.503964 | 0.520161 |
| raw10 | dynamic,1000 | 0.801739 | 0.307782 | 0.299683 | 0.315126 | 0.315018 | 0.495535 | 0.488004 | 0.507819 | 0.743713 |

Code:

Again, there are two versions: the inline version and the parallel version. badCar.c is the inline version, and can be compiled with “gcc badCar.c”. badCarParallelParked.c is the parallel version, and can be compiled with “gcc -openmp -o badcar badCarParallelParked.c”

Both programs required you to specify the number of iterations. The Parallel

one requires an extra number for the number of threads. Below are the two execution

commands with the about compilation used:

inline:

“./a.out [iterations]”

parallel

“./badcar [iterations] [number of threads]”

When running the parallel version, you can create an environment variable called

“OMP\_SCHEDULE” to specify it's schedule. This can be done with:

“export OMP\_SCHEDULE="[the schedule goes here]"”