Performance Evaluation

Introduction:

For each sample size n, the SAR of nC2 interpolated lines was computed with the use of the parallelized map function. Since nC2 is about n^2 and the SAR function is O(n), the time complexity of the map operation was O(n^3).

The L1 was then found by minimizing the SAR with the parallelized fold function.

Each test was run with 1, 4, and 8 threads in turn. The tests are run from dataPar.c in main.

Hardware:

Intel® Core™ i7-3520M CPU @ 2.90GHz × 4
9.6 GiB RAM

Task 1: CPI Data

These all produced the same values as the table in the paper, they all ran in 0 or 1 ms, boooriiing, skip ahead.

Output:

```
computing L1 of 6 points
with 0 threads
Map: 0 ms
fold1: 0 ms
L1-> intercept:87.599998,  slope:1.325001,  SAR:1.624992

with 4 threads
Map: 0 ms
fold1: 0 ms
L1-> intercept:87.599998,  slope:1.325001,  SAR:1.624992

with 8 threads
Map: 1 ms
fold1: 0 ms
L1-> intercept:87.599998,  slope:1.325001,  SAR:1.624992

------------------------

computing L1 of 10 points
with 0 threads
Map: 0 ms
fold1: 0 ms
L1-> intercept:86.560005,  slope:1.920000,  SAR:6.599991

with 4 threads
Map: 1 ms
fold1: 0 ms
L1-> intercept:86.560005,  slope:1.920000,  SAR:6.599991

with 8 threads
Map: 0 ms
fold1: 0 ms
L1-> intercept:86.560005,  slope:1.920000,  SAR:6.599991
```

```
------------------------



computing L1 of 14 points
with 0 threads
Map: 1 ms
fold1: 0 ms
L1-> intercept:84.571434,  slope:2.242857,  SAR:8.857140

with 4 threads
Map: 0 ms
fold1: 0 ms
L1-> intercept:84.571434,  slope:2.242857,  SAR:8.857140

with 8 threads
Map: 0 ms
fold1: 1 ms
L1-> intercept:84.571434,  slope:2.242857,  SAR:8.857140

------------------------

computing L1 of 18 points
with 0 threads
Map: 0 ms
fold1: 0 ms
L1-> intercept:86.000000,  slope:2.100000,  SAR:11.300003

with 4 threads
Map: 0 ms
fold1: 0 ms
L1-> intercept:86.000000,  slope:2.100000,  SAR:11.300003

with 8 threads
Map: 1 ms
fold1: 0 ms
L1-> intercept:86.000000,  slope:2.100000,  SAR:11.300003

------------------------
```

Task 2: Stremflow Data

n = 356:
With O(356^3) we can now see significant speedups from multi-threading, in map at least. Going going from single to 4-threaded halves the computation time! However, adding another 4 threads doesn't seem to affect performance at all (the difference between 93 and 89 ms is just random noise). I will come back to this in the "More Threads than Cores" section.

And Fold1 still takes a negligible amount of time.

Output:
```
computing L1 of 365 points
with 0 threads
Map: 189 ms
fold1: 1 ms
L1-> intercept:173.590912,  slope:-0.116162,  SAR:90229.960938

with 4 threads
Map: 89 ms
fold1: 1 ms
L1-> intercept:173.590912,  slope:-0.116162,  SAR:90229.960938

with 8 threads
Map: 93 ms
fold1: 0 ms
L1-> intercept:173.590912,  slope:-0.116162,  SAR:90229.960938

------------------------
```

n = 3652:
For this huge computation we see the same behaviour from map: the jump to 4 threads halves the time, but increasing to 8 does nothing.

As for fold1, we can see that for this application the reduction step needn't be parallelized at all. No matter the number of threads, the time for this calculation usually hovered at around 55 ms.

Output:
```
computing L1 of 3652 points
with 0 threads
Map: 182419 ms
fold1: 49 ms
L1-> intercept:137.252686,  slope:-0.002679,  SAR:862936.937500

with 4 threads
Map: 86627 ms
fold1: 60 ms
L1-> intercept:137.252686,  slope:-0.002679,  SAR:862936.937500

with 8 threads
Map: 86296 ms
fold1: 51 ms
L1-> intercept:137.252686,  slope:-0.002679,  SAR:862936.937500

------------------------
```

More Threads than Cores

So does spawning more threads than you have cores help performance or not?

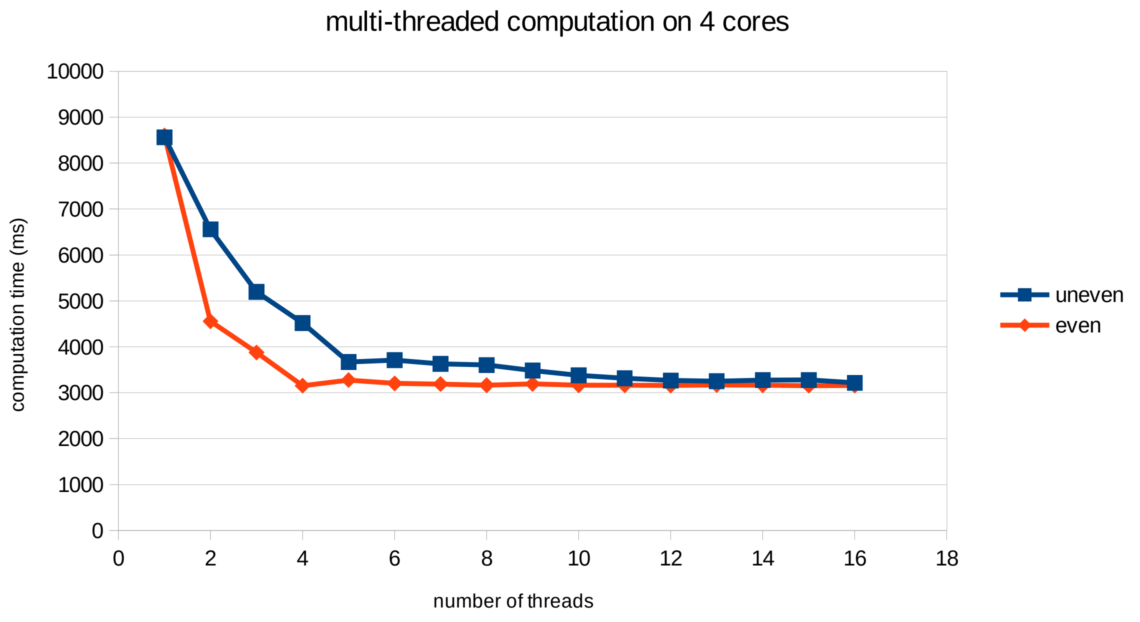
I hypothesize that when there are more threads than cores, the overall computation time is cut down if the cores that finish their thread first can start on another thread. For example, let's say you have 3 threads that take 1, 1, and 2 seconds respectively. If you have 2 cores working on the problem, then one core can run threads 1 and 2 sequentially while the other core runs thread 3. However, if all your threads take the same time to compute, then you don't need any more threads than you have cores, by the hypothesis.

To test this, I made a function with parameters acc and num, which just loops num times. I then made 2 arrays of length 10^5; One had elements increasing from 1 to 10^5, and the other had (10^5)/2 in every spot. I then folded over both arrays in turn and timed the results (I also wanted to prove to myself that fold1 was properly parallelized, since the main tests were so unsatisfactory).

From the output and corresponding chart below you can see that with the rising array the time interval kept decreasing and only hit its minimum level at about 11 threads, whereas with the flat array it stopped decreasing at 4. This supports the hypothesis. The fact that task 2 didn't benefit much from extra cores suggests that each chunk was a roughly equal amount of work, which makes sense given the SAR function.

My conclusion is that if you suspect that the workload is going to be unevenly distributed it is definitely worth spawning more threads than you have cores.

You can find the code for this test in “fold1.c”



output:

```
extra test!
commencing O(n^2) operation on 100000 elements

first with workload distributed unevenly over threads:
8561 ms with 1 threads
6557 ms with 2 threads
5197 ms with 3 threads
4518 ms with 4 threads
3669 ms with 5 threads
```

```
3709 ms with 6 threads
3628 ms with 7 threads
3604 ms with 8 threads
3484 ms with 9 threads
3380 ms with 10 threads
3314 ms with 11 threads
3266 ms with 12 threads
3252 ms with 13 threads
3276 ms with 14 threads
3279 ms with 15 threads
3215 ms with 16 threads

now with workload distributed evenly over threads:
8601 ms with 1 threads
4555 ms with 2 threads
3876 ms with 3 threads
3154 ms with 4 threads
3276 ms with 5 threads
3203 ms with 6 threads
3189 ms with 7 threads
3164 ms with 8 threads
3190 ms with 9 threads
3163 ms with 10 threads
3165 ms with 11 threads
3157 ms with 12 threads
3169 ms with 13 threads
3162 ms with 14 threads
3154 ms with 15 threads
3158 ms with 16 threads
```