Braden Ackles

CS 475

Project #3

5/1/2016

# Machine

The machine I ran this on was FLIP3. At the time of running it had a usage under 5% in all three categories.

# Table

NOTE: for Tables the X axis is number of threads and the Y axis is NUM(Padding). All compiler optimizations are disabled.

MegaCalcs per second

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Fix #2: 1** | **Fix #2: 2** | **Fix #2: 4** | **Fix #1: 1** | **Fix #1: 2** | **Fix #1: 4** |
| 0 | 303.563 | 606.206 | 1164.51 | 292.163 | 557.013 | 419.709 |
| 1 | 304.395 | 586.78 | 1166.39 | 291.968 | 308.937 | 324.098 |
| 2 | 299.767 | 608.493 | 1109.99 | 295.454 | 383.294 | 352.374 |
| 3 | 286.726 | 576.716 | 1168.33 | 292.461 | 467.975 | 465.079 |
| 4 | 299.519 | 583.184 | 1167.29 | 248.333 | 268.542 | 326.188 |
| 5 | 293.523 | 592.206 | 1144.07 | 294.532 | 543.632 | 421.734 |
| 6 | 294.842 | 608.227 | 1167.3 | 299.651 | 560.374 | 381.396 |
| 7 | 299.189 | 606.985 | 1163.82 | 298.877 | 610.108 | 402.326 |
| 8 | 296.676 | 583.322 | 1165.73 | 259.33 | 516.395 | 400.633 |
| 9 | 291.685 | 583.487 | 1166.27 | 251.909 | 516.319 | 430.126 |
| 10 | 303.796 | 583.175 | 1149.14 | 229.425 | 389.712 | 396.864 |
| 11 | 290.31 | 583.389 | 1165.97 | 295.739 | 580.691 | 695.625 |
| 12 | 304.131 | 607.015 | 1166.29 | 280.362 | 579.602 | 483.218 |
| 13 | 298.314 | 594.384 | 1166.48 | 300.953 | 583.782 | 484.14 |
| 14 | 299.202 | 584.456 | 1164.03 | 258.598 | 502.956 | 747.368 |
| 15 | 290.106 | 583.485 | 1165.58 | 290.494 | 584.36 | 1167.95 |
| 16 | 297.341 | 583.522 | 1166.01 | 248.731 | 485.427 | 1008.09 |

Times speedup compared to T1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Fix #2: 1** | **Fix #2: 2** | **Fix #2: 4** | **Fix #1: 1** | **Fix #1: 2** | **Fix #1: 4** |
| 0 | 1 | 1.94588 | 3.66187 | 1 | 1.58839 | 1.03117 |
| 1 | 1 | 1.93956 | 3.65386 | 1 | 1.55515 | 1.81771 |
| 2 | 1 | 1.93826 | 3.61043 | 1 | 0.888108 | 1.21149 |
| 3 | 1 | 2.01997 | 3.87402 | 1 | 1.77517 | 1.20052 |
| 4 | 1 | 1.94034 | 3.87709 | 1 | 1.66211 | 1.27037 |
| 5 | 1 | 1.91965 | 3.83947 | 1 | 1.48541 | 1.28799 |
| 6 | 1 | 2.13265 | 4.26866 | 1 | 1.55485 | 2.02069 |
| 7 | 1 | 1.99826 | 3.99473 | 1 | 1.99967 | 2.05505 |
| 8 | 1 | 1.99735 | 3.99419 | 1 | 1.99582 | 1.96613 |
| 9 | 1 | 1.99976 | 3.99739 | 1 | 2.03235 | 1.9345 |
| 10 | 1 | 2.03715 | 3.98467 | 1 | 1.92328 | 1.9264 |
| 11 | 1 | 1.97317 | 3.93779 | 1 | 2.03136 | 2.58444 |
| 12 | 1 | 1.9922 | 3.96278 | 1 | 1.95949 | 2.34794 |
| 13 | 1 | 1.97651 | 3.83887 | 1 | 1.9987 | 1.84958 |
| 14 | 1 | 1.91905 | 3.84014 | 1 | 2.00613 | 2.5481 |
| 15 | 1 | 1.95082 | 3.89998 | 1 | 2.00063 | 3.96312 |
| 16 | 1 | 1.96994 | 3.8432 | 1 | 1.99411 | 3.99378 |
|  |  |  |  |  |  |  |

# Graphs

MegaCalcs Per second

Times speed up (T1/TN)

# Analysis

In the analysis of the output megacalcs and times increased We can see that fix number two has no increase or decrease in performance in either chart. This is because it is consistent and doesn’t rely on padding in the sense of adding to the cache line when writing but by making a private version of the variable on each of the threads stacks. This means that is gets consistent performance at the max possible for that number of threads, this is because since each thread has its own copy it can read and write it and not have to worry about any other thread trying to interact with that data like it does in fix number 1. The Yellow, Orange, and Grey lines in the second graph all represent fix number two.

Fix number one on the other hand increases slowly as the padding increases. But increases more when the amount of padding plus the integer add to be a complete cache line because then it is all on one line. This is seen at 15 and 16 byte(padding) because 15+1 bytes is 16 which is one fourth the length of a cache line so it is easy to pull it in and perform operations on. But as the NUM increases the times increase does grow except on 1 core because it wouldn’t make any sense for there to be any speed up because it would be (T1/T1) which would just be one.

So to conclude, Fix #2 is much more efficient and consistent. It basically provides the speeds at constant rates. This is because each of the cores in fix #2 gets a private copy of the variable. Where in fix #1 there is not private so we have to read in cache lines and provides the ability for two cores to false share or go to read or write data at the same time. This causes the smaller increase but as we increase the size on the cache line we can read it in in whole.