Mitigating False Sharing in Distributed Shared Memory

UP850874

Computer Science

1. Abstract

This poster explores methods of reducing false sharing in distributed shared memory. Using simulations, we can first identify how false sharing works and then how implementing different padding sizes can help to mitigate the chance of false sharing occurring.

2. Introduction

The intent of the poster is to explore false sharing remedies within distributed shared memory and identify a plausible method to lessen its effects on the system.

False sharing is a common problem in distributed systems. It is where two or more processes share parts of a page, but only one can fully access each part or variables within. This can cause an increase in cache misses where the process goes to main memory to collect the value, which subsequently will negatively affect the performance of the distributed system.

The key method researched into is called padding. This is when programs will 'pad' their variables so only one will reside within a single line of memory.

Constraints to this research are resources and time, as some of the other methods identified are more fitting but require more complex tools and in-depth knowledge to implement.

3. Problem Statement/Aim & Objectives

The problem being solved is the impact that false sharing has on a distributed system. It is important because the larger a page that is utilised in distributed shared memory, the more likely that an event of false sharing will happen.

The main aim is to create a working method to counter false sharing and therefore increase the performance within distributed shared memory. This will be done by analysing a false sharing scenario, determining a solution, then testing it to identify how impactful the designed solution is.

How frequently false sharing occurs (in seconds) will be measured in a simulated distributed system and then how much that changes when implementing a potential solution.

4. Background

There are a few different ways to combat false sharing which is discussed in the literature surrounding this topic. To narrow the scope, this poster concentrates on measures taken to reduce false sharing rather than preventative measures, such as those discussed by Oracle, of which their leading point is to make use of private data as much as possible and optimise how data is that allocated. (6.2 False Sharing And How To Avoid It (Sun Studio 12: OpenMP API User's Guide), n.d.).

The first approach is to use padding (alongside spacing in this example) to fill out pages in the cache so that only a single variable resides in that memory location, thus preventing any invalidation of that page when multiple threads read/write to the data. (Sae-eung, 2010). The drawback of this is you sacrifice memory space when you increase the data set size with this padding (Sae-eung, 2010). Fig.1 shows the difference when using padding/spacing on an array.

A second approach is called grouping. This is when programs will group different data from the pages together so the data structures in memory more closely align. This transformation also improves spatial locality and therefore improves performance in a distributed system (Jeremiassen & Eggers, n.d.). Fig.2 demonstrates how this works and shows why it's a good alternative to padding.

The last important approach is the hybrid between padding and grouping. A well tested conclusion is that the results of using this hybrid supports the use of adding padding to decrease instances of false sharing (Measuring the Impact of False Sharing, n.d.). See fig.3 for more detail on the comparison.

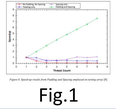


Fig.1

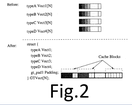


Fig.2

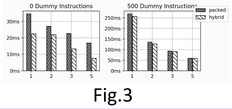
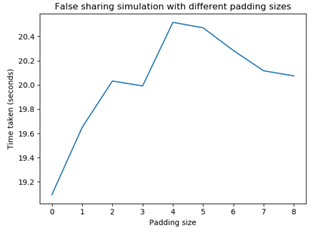


Fig.3

5 . Experimental Design

From our research we can tell that padding is influential in decreasing the amount of false sharing in a distributed system. We shall be identifying the optimal amount of padding to achieve our aim in this experiment. The python programming language is used alongside time, threading, and matplotlib libraries to simulate a scenario where false sharing can happen. We use a fixed sized test array, then two concurrent threads to access adjacent array elements, which can result in our false sharing incident. We time how long each incident takes to occur with different sizes of padding incrementing from 0 to 8 bytes. The simulation is run once without padding in the array to establish a baseline time, then it is repeated multiple times with an increased padding length. Finally, we create a clear graph of padding size over false sharing event time.
Data is collected by print statements to the python console and through the generation of the line graph.
These steps allow us to compare the time taken to complete the false sharing simulation with different padding sizes and to observe how the simulation time changes accordingly. By plotting the results on a line graph, we can visualise the relationship between padding size and the simulation time to gain a better insight into the impact that padding presents to mitigating false sharing.

6. Results




```
padding of size 0 : 19.09394907951355 seconds
padding of size 1 : 19.648449897766113 seconds
padding of size 2 : 20.031824588775635 seconds
padding of size 3 : 19.991456747055054 seconds
padding of size 4 : 20.515793323516846 seconds
padding of size 5 : 20.470721006393433 seconds
padding of size 6 : 20.283387660980225 seconds
padding of size 7 : 20.116329431533813 seconds
padding of size 8 : 20.074196577072144 seconds
```

The results are split into a line graph and a console output. This is to maintain the accuracy of what transpired and to provide a visual aide to display the relationship between padding and false sharing.

9 simulations were conducted, and we can see a clear correlation in our first five simulations whereby when padding size increases, so does the time before false sharing occurs. The baseline result with zero padding takes 19.09 seconds but with padding we see about a 0.4 second solid increase in time. This continues all the way to 4 bytes of padding where this trend peaks at 20.52 seconds. We can identify a potential anomaly result which is the third byte of padding that doesn't quite follow the trend. Therefore, we can deduce that this is caused by false sharing being unpredictable and increases the amount of uncertainty within each gathered result - this result might not be as valid as the others because of this fluctuation.

After 4 bytes of padding, we see a decreasing trend in the amount of time before false sharing occurs and this only worsens when more padding is used. In summary, we can identify that 4 bytes of padding is the optimal amount from this experiment.

7. Discussion

A sequential approach was used to analyse the results. The outcome is that 4 bytes is the optimal amount of padding when testing between the range of 0 to 8, this is shown by the 4 bytes generating the highest time of 20.52 seconds before false sharing occurred. This experiment is limited by time and resources for failing to increase the range of padding sizes above 8 bytes, overall reducing its potential to find the correct amount.

The outcome was unexpected it was assumed to be a proportional increase between time and padding, which was proved to not be the case. Unsure of this experiments result when starting, and therefore removing a potential for a biased result, this experiment provides an interesting inspection into padding and identifies a set number of bytes of padding to improve performance when without the ability to adapt to the specifics of the system and distributed shared memory.

There is also the possibility of inaccuracies being created in this simulation due to its setup. Python isn't specific to running system simulations even though a plausible replication was attempted. In a real distributed system, there would be far more threads accessing the same data and more arrays and variables being read/written to. It is still a good first step however to uncovering an impactful solution to mitigating false sharing.

8. Conclusions

After researching and experimenting with padding to mitigate false sharing, we can conclude that padding does have a positive impact upon increasing the amount of time before false sharing happens, therefore increasing the performance of the system.

We have narrowed down that 4 bytes of padding is optimal (when testing between 0 and 8 bytes) by providing a 1.42189 second improvement time. The peak result was 4 bytes of padding taking 20.5 seconds to occur compared to our baseline of 0 bytes of padding taking 19.1 seconds.

Overall, we have proven a working method to reduce false sharing by recording a simulation of false sharing occurring and the impact of using different padding sizes to mitigate the frequency of false sharing happening.

9. Future Work

There are lots of limitations to this original experiment that can be changed to create a more legitimate and trustworthy conclusion. In the future, the experiment can be changed in the following ways:

1. Increasing the range of padding sizes up to 64 bytes will allow a more concrete result that will show the best padding size.
2. Increase the total amount of threads to a more reasonable number to better simulate a distributed shared memory.
3. Complete the experiment multiple times to allow averages to be deduced and consequentially get more accurate results.
4. Use a more dedicated software to better simulate false sharing in a distributed system.

10. References

- 6.2 False Sharing And How To Avoid It (Sun Studio 12: OpenMP API User's Guide). (n.d.). https://docs.oracle.com/cd/E19205-01/819-5270/6n7c71veg/index.html
- Sae-eung, S. (2010). Analysis of False Cache Line Sharing Effects on Multicore CPUs [San Jose State University]. https://doi.org/10.31979/etd.bv2q-hd7t
- Jeremiassen, T. E., & Eggers, S. J. (n.d.). Reducing False Sharing on Shared Memory Multiprocessors through Compile Time Data Transformations.
- Measuring the Impact of False Sharing. (n.d.). https://alic.dev/blog/false-sharing.html