Braden Ackles

Project 6

CS 375

5/25/2016

So I have been having trouble with visual studios and have sadly spent overly too much time trying to get visual studios to work that I didn't focus as much on the assignment I could not get openmp to work on visual studios on my laptop or desktop. I also tried to use rabbit and was never able to log in it always said some form of access denied whenever I tried to ssh to it. So that being said I don't have any actual data to use for graphs or for tables. But I still wanted to submit the assignment before my 2 bonus days for the assignment are used. I understand that it is my fault for waiting till last minute to do the assignment and if when you read this you would like to help me get it up and running so I can actually see the real results please email me at ackles@oregonstate.edu. However, I would like to explain what I think the graphs would look like and then answer the questions from there.

Machine

I was unable to get it working as previously said, But I tried visual studios on my laptop and desktop as well as time reserved on rabbit.

Multiply Arrays

Global Work size

The graph for the multiply would increase extremely quickly at the beginning as you increase the global work size. However, after it grows to its peak it will top off and become a flat line. This could be because there are more items to do in parallel and can save ever increasing amounts of time.

Local Work Size

This is more dependent on the GPU that you will be using. If you have intel integrated graphics having a work group of 32 might be too big and actually cause a decrease in speed compared to a work group on 12 (a number more reasonable to that specific GPU). However, if you run it on a nicer GPU that you might see a very large increase because you are increasing the amount of parallelism at once. So a work group of 32 will have many more giga-numbers calculated per second because more numbers can be done in parallel. Although there is definitely the sweet spot and there is still an amazing performance increase with oversized local work group sizes the more overly sized you get theoretically there will be less of a gain from doing it.

So the patterns we could expect to see is a sweet spot in both global and local work group size. Global as you increase should almost always help because there are more items to save time in calculating in parallel. The local work group size has a more noticeable sweet spot that is dependent on the GPU and its amount of parallelism. If it can do more items in parallel, then a bigger work group size will show and increase where if you exceed the work group size for the amount of data that can be done in parallel you will see less of an increase. Although you will still see an increase because people who design these things are smart and have basically a manager that throws data to a GPU core to calculate on when one becomes free. So you will still see an increase but if you can break your problem up in to as many problems as the GPU has cores then you will see maximum performance.

Multiply Arrays plus ADD

This paragraph will be pretty short because when it comes to adding at the end its basically free. The multiply takes of the calculation time and then the add is basically free of charge for time to calculate. So because of this it should be very similar to the previous multiply array information.

Reduction

Reduction is done by a single work group.

Reduction will behave differently than the rest. This is because it will grow fast but also peak out much faster this is because as it goes through the reduction the problem gets smaller and smaller and there for fewer and fewer cores can perform parallelism to tackle the problem. This means that it will grow very fast at the beginning but then peak out and then no more increase. This would show up as a flat line on the graph after the increase had happened. Since reduction is done by a single work group and each task is done by a single thread this means that as more tasks are completed there are that many less threads performing parallelism at that time and there for aren't as fast as if all threads were still going. This also means there is a loss of compute power because all threads aren't busy after the first set is performed and every time after that.

Proper GPU Programming

The best way to program in parallel is to split the problem in to as many parallel problems as possible. If there are no dependencies on anything else, then it should be done in parallel if there are enough of them that way the time is saved. That being said Multiply arrays and multiply arrays and add is good to do in parallel because there are no dependencies on any other positions in the array. The data is consecutive and can be streamed to the GPU. Reduction is still good to do in parallel but its diminishing so the problems should be larger to get more time saved. Because then more data at the beginning can be paralleled however in the end it will still be basically the same proportion of time saved.

Conclusion

As I stated in the introduction to this paper please contact me if you so graciously would like to help me figure out my problem and get my code up and working. Thank you for understanding the circumstances. I tried my best to understand the material and explain it even without tables and graphs. I look forward to hearing from you! ☺