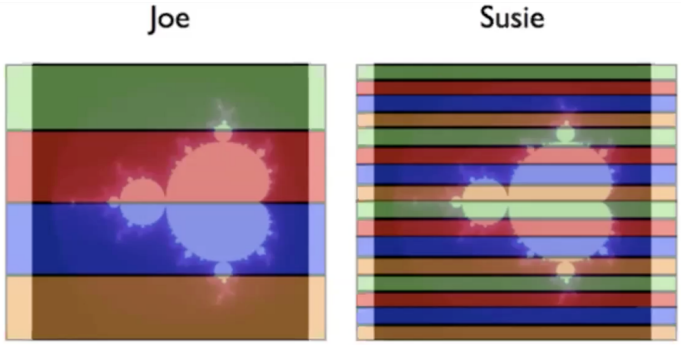
HW2 - Zeyu Zhang

Theoretical Analysis

Theoretically, Susie's approach is better and she is the one that is more qualified for a full-time job because it provides better load balancing performance.

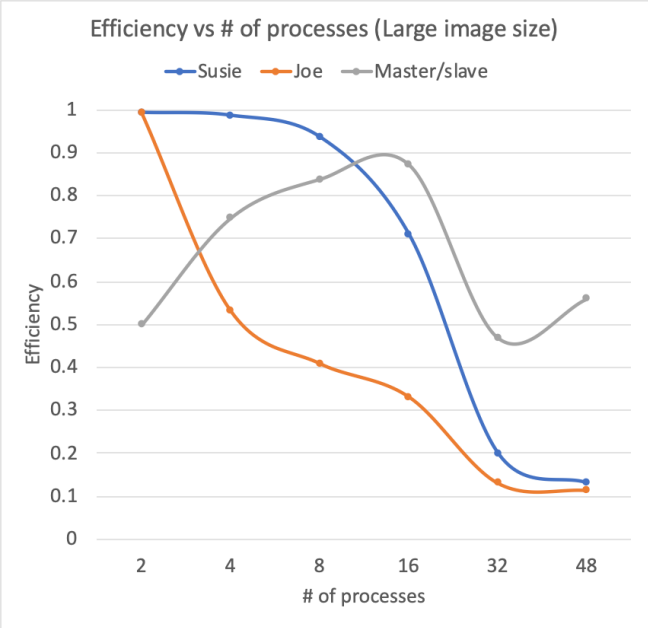
A lot of image, including the Mandelbrot set image, is somehow continuous rather than discrete in space. In other word, in some areas of the image, there are a lot of shapes to render while in some other areas, the image is empty. As a result, it is a relatively bad strategy to divide the image into several blocks with continuous row (or column) number because some blocks may contains no content while some others may contains a lot of information. Susie's strategy is much better because a job contains of rows that come from everywhere of the image, from the top to the bottom.

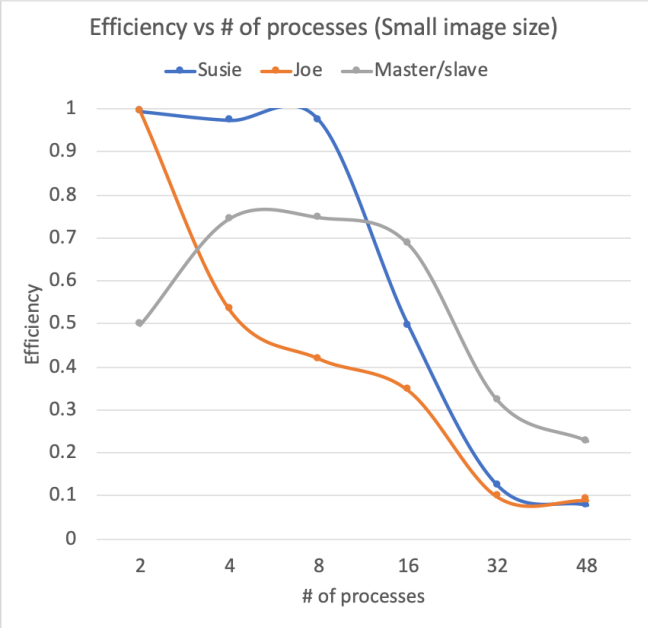


As illustrated in the lecture video, every pixel within the white part in the image requires max (511) iterations while the pixels in the black part require few iterations. in Joe's strategy, the two jobs in the middle are much heavier than the two on the top and bottom. In contrast, in Susie's strategy, all four processes cooperation on both computational-cheap area and computational-heavy area of the image.

Real Data Analysis

Parallel efficiency against # of processes





As is shown in the figures, no matter for large or small image, Susie's strategy has better parallel efficiency than Joe's. Obviously these two strategy has same work and total massage passing overhead. Hence, the better efficiency comes from the better load balancing performance.

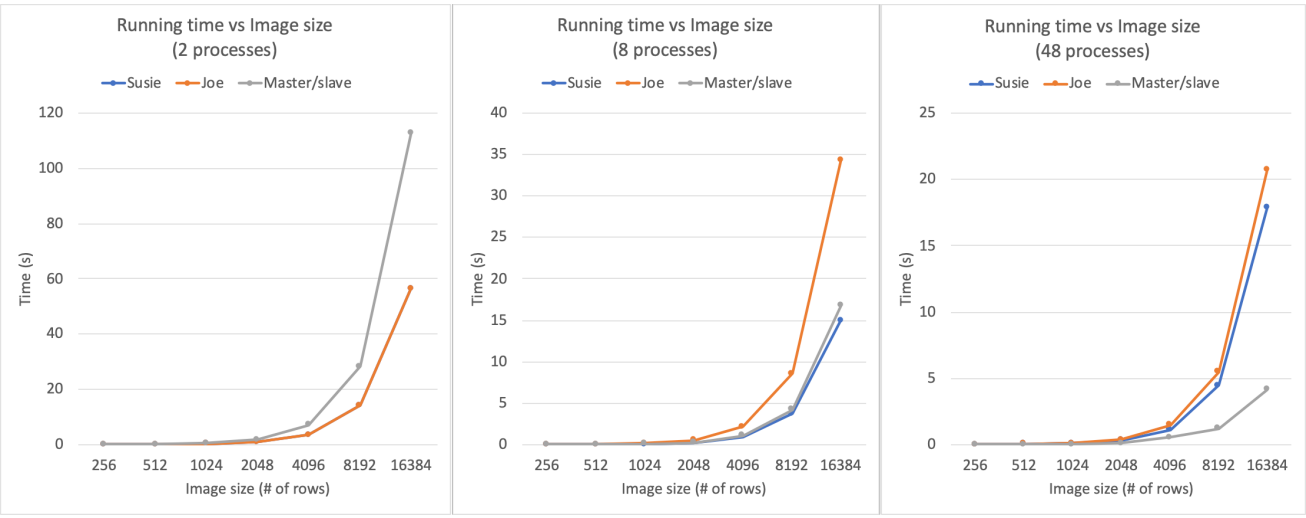
All three strategies shows a dramatic efficiency drop at 16 processes because $class16$ queue has 12 cores per node. With more than 12 processes (1 process/node), we have to do inter-core communication which is much slower than the communication within a core.

Noted that when # of processes is very large, message passing overhead dominates in the total running time because we need to send more small messages instead of few large messages. Hence, as is illustrated in the figures, Joe's and Susie's strategy shows similar efficiency when # of processes is large.

For the master/slave model, the efficiency firstly increase and then decrease as the # of processes grows. This is because the core idea of this model is to achieve extreme balance load at the cost of the computation resource of the master process. If we have only a few processes, this sacrifice is huge. This is why when we have only 2 processes, the efficiency is less than 0.5: We have two processes while only one is doing computation in a serial manner. However, this sacrifice could be hidden when we have a lot of processes. As is shown in the left figure, when we have 48 processes, thanks to the excellent load balancing performance, master/slave model provide much higher efficiency compared to Joe's and Susie's strategy.

Running time against image size

In this test, I use square image (# of rows = # of column) and keep doubling the # of rows to see how these three approaches behave. These figures provide another view of the conclusions above.



Susie's strategy provides better load balancing performance and is faster than Joe's no matter how many processes we have. The difference between these two strategies becomes larger as the image size grows.

In terms of the master/slave model, when we have a few processes, the large overhead of the master processes dominates and make it the slowest one. However, as the # of processes grows, the sacrifice of the master process is mitigated. Its performance improves continuously and finally become the best approach among three. It's superiority becomes extremely obvious when we have a very large image to calculate. In this case, the (original) tiny job unbalancing in Joe's and Susie's strategy is enlarged due to the increasing # of column while master/slave model always provide perfect load balancing performance.

Conclusions

Susie's load balancing strategy is better than Joe's because it scatters a job's rows in different location of the image. However, both strategies are kind of naive compared to master/slave model. Even though this model has some overhead in terms of the master process and it requires more massage passing than naive models, the perfect load balancing performance makes it the optimal one (the most scalable one) for very large image.