MPI implementation of the parallel fast marching method

Wisart Thongyoy

ABSTRACT

In this work, we implement the parallel fast marching method in order to speed up the calculation time of finding travel time. We also optimize the implementation of the fast marching method by allowing repeated element in the binary heap. Using element method, the calculation time is up to 100 times faster. The parallelization result shown the success of the method, but the speed up factor was dropped when using many processing cores.

INTRODUCTION

The parallel method in this work adopted from the method of Yang and Stern (2017) which uses a domain decomposition method. Yang and Stern (2017) suggested that instead of updating the minimum time point in the domain, the process can be parallelized by decomposing the domain and update minimum time point in each subdomain. Since each subdomain run separately, it can affect the continuity of the wavefront between neighboring subdomains. To alleviate this, instead of continuously propagate wavefront in each subdomain separately, the subdomains can only be updated to a certain point of time and must wait until all of the subdomains reached the same point, after that, a new limiting time point is set and the process is repeated. This will be clarified and the pseudocode will be given in the next section.

THEORY AND METHODS

Sequential fast marching method

The fast marching method was developed to solve the Eikonal equation:

$$|\nabla\psi(\mathbf{x})|F(\mathbf{x}) = 1,\ \mathbf{x} \in \Omega, \tag{1}$$

where $\psi(\mathbf{x})$ is travel time, Ω is the domain, and $F(\mathbf{x})$ is a positive speed function. In geophysics, the equation is generally used to calculate travel time from source to any point in the domain.

Eq. 1 can be solved numerically by discretizing the domain. After that, use the Godunov-type finite difference approximation, suppose that our domain is in the 3-dimensional space, the result is

$$[\max(D^{-x}_{i,j,k}\psi, -D^{+x}_{i,j,k}, 0)^2 + \max(D^{-y}_{i,j,k}\psi, -D^{+y}_{i,j,k}, 0)^2 + \max(D^{-z}_{i,j,k}\psi, -D^{+z}_{i,j,k}, 0)^2]^{1/2} = \frac{1}{F_{i,j,k}}, \tag{2}$$

where $D^{-x}_{i,j,k}\psi$ and $D^{+x}_{i,j,k}\psi$ are backward and forward finite difference approximations of the spatial derivative $\dfrac{\partial\psi}{\partial x}$, respectively. For the first-order scheme, which is used in this work, the approximations are

$$D^{-x}_{i,j,k}\psi = \frac{\psi_{i,j,k} - \psi_{i-1,j,k}}{\Delta x}, \quad D^{+x}_{i,j,k}\psi = \frac{\psi_{i+1,j,k} - \psi_{i,j,k}}{\Delta x}. \tag{3}$$

Note that, this rule can also be applied similarly along the y and z directions.

Algorithm 1 is the serial procedure of using the Godunov approximation in the fast marching method. In fast marching method, each point in the domain has status represented

Algorithm 1 Sequential fast marching method

```
ψ ← +∞
G ← Downwind
ψ(x_s) ← ψ_0,  x_s are source positions
G(x_s) ← Upwind
G(N(x_s)) ← Band, N(x_s) are neighbors of x_s
ψ(N(x_s)) ← solve Godunov
while Band is not empty
    x_m ← x : gives min(ψ(x)), x ∈ Band
    G(x_m) ← Upwind
    ψ(x_m) ← solve Godunov
    for all G(N(x_m)) is not Upwind
        G(N(x_m)) ← Band
        ψ(N(x_m)) ← solve Godunov
    end for
end while
```

by tag(G). There are three types of tag: 1. Upwind node, whose travel time at that point was calculated and will not be changed, 2. Band node, whose travel time was calculated but can still be changed, and 3. Downwind node, whose travel time is not calculated, normally the traveltime of points with unknown travel time are set to be infinity, practically a very large number. Fast marching method mimics the behavior of wave front expanding, which means that the travel time will be calculated first at the source point which, certainly, has the least traveltime. Then, its neighbors is updated and represent as band node. The process is repeatd by selecting the least traveltime node to be updated until reached the domain limit.

The method starts with assigning infinity to every point and tagging them to be Downwind. Next, the travel time at source point(s) is set and tag to be Upwind, note that the travel time of the source point can be set to be any value, but, in geophysics, since it mimics the traveling of wave, the travel time here is set to be zero. Then, the neighbors of the source point(s) are tag to Band and the travel times are assigned by solving equation 2. After that, the point with the least traveltime in the band is picked, tag to band, and updated travel time. Finally, the neighbors of the minimum travel time point is updated and tag to be Band if its tag is not Upwind. The process is repeated until there is no point with Band tag.

Speed up the fast marching method

Finding the minimum time in the band can be very slow, so the binary heap algorithm is generally used to speed up the process. The heap stores values of both $\mathbf{x}$ and $\psi(\mathbf{x})$. The algorithm of adding a new point to the heap is shown in **Algorithm 2**.

Algorithm 2 Adding point to heap

1: $\mathbf{x}_a \leftarrow$ point to be added
2: if $G(\mathbf{x}_a)$ is Downwind
3: add $\mathbf{x}_a$ and $\psi(\mathbf{x}_a)$ to heap
4: else if $G(\mathbf{x}_a)$ is Downwind
5: Find old $\mathbf{x}_a$ and $\psi(\mathbf{x}_a)$, from heap
6: replace old value with $\mathbf{x}_a$ and $\psi(\mathbf{x}_a)$
7: end if

When we add a new point to the heap, it is possible that the point is already in the heap (the point is in the heap and then updated to be a new value). If this happens, we have to replace the old value in the heap to make sure that the point will not be updated incorrectly, twice. The deleting from heap process is used to make sure that there are no repeated elements in the heap. However, we found that the finding old point in the heap to delete (5th line in **Algorithm 2**) is very expensive: it can consume about 80-90% of the total calculation time. To overcome this, we allow the heap to have repeated elements and add more conditions. There are two possible values in the heap: $\mathbf{x}$ and $\psi(\mathbf{x})$, since when the element is repeated the only quantity that is repeated is only $\mathbf{x}$, for example, we can have $[(\mathbf{x}, \psi_{old}(\mathbf{x})), (\mathbf{x}, \psi_{new}(\mathbf{x}))]$ in the heap. Now, if we have to find and delete the minimum value in the heap, we have to check whether or not the travel time in the heap and in the domain are equal or not, if not we skip the value. This is done based on the fact that the newest travel time added to the heap must be equal to the present travel time in the domain.

Parallelization by domain decompostion

To ensure the continuity of the wavefront, Yang and Stern (2017) used overlapping domain decomposition as shown in Figure 1.

Algorithm 3 Parallel fast marching by domain decomposition

1: while all points are not Upwind
2: mintime $\leftarrow$ minimum time in heap
3: global mintime $\leftarrow$ All reduce(mintime)
4: time limit $\leftarrow$ global mintime + stride
5: while mintime $<$ timelimit AND band size != 0
6: mintime $\leftarrow$ minimum time in heap
7: Genenral fast marching method
8: end while
9: the domain synchronization
10: end while

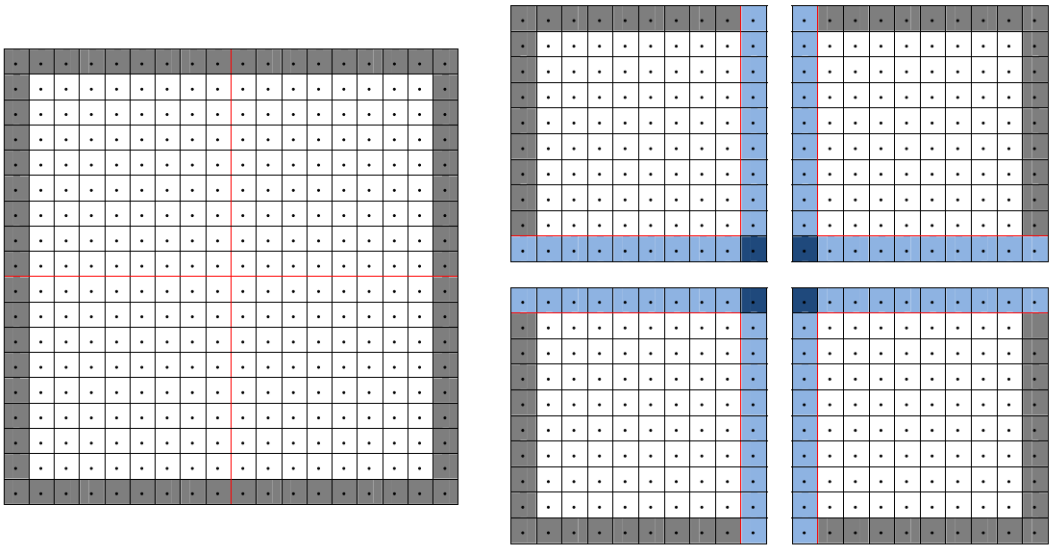

Figure 1: Domain decomposition in parallel computation(after Yang and Stern (2017))

Algorithm 3 shows the pseudo code of parallel fast marching method, the stride value is the time step that the wave front will be advanced. Yang and Stern (2017), suggest to use stride equal to $2\Delta x$, In this work, we also used this value.

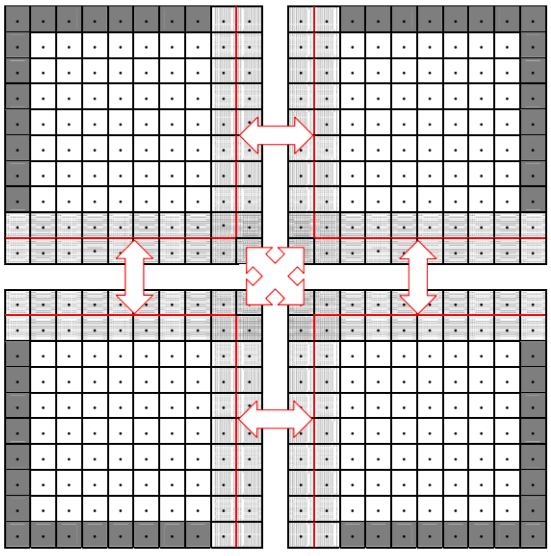

Figure 2: Data exchange between neighboring processes(after Yang and Stern (2017))

The domain synchronization is done as overlapping domain decomposition. The overlapping part shown in Figure 2 will set the new value after synchronization as minimum time. For example, if there are points $\mathbf{x}$ in 4 different sub domain with travel time t_1, t_2, t_3, and t_4, after synchronization all of the time value at point $\mathbf{x}$ are set to be $\min(t_1, t_2, t_3, t_4)$. To speed up the synchronization process, the process is performed selectively, that is, not

all of the points in the overlapping region are sent to neighboring processes but only points that are updated or the traveltimes are changed.

Testing model and machine

To test the performance of the code, we set up testing models. The testing models are cubic with a single point source at the center of the domain. The size of the domain will be varied. In this work, we use variable nh to indicate the size of the domain, where nh is number of points in each direction or nh^3 is amount of total grid points in the domain. The testing machine is a shared-memory system server with 24 Intel Xeon 2.5GHz cpus.

RESULTS

Before and after allowing repeated element

Figure 3 shows the time of before and after allowing repeated element. It is clear that the calculation decreases dramatically. This is because our added condition has much less calculation time than searching element in the heap. Note that Figure 3 is using old version of our program where synchronization is done by exchanging every points in the overlapping region between neighboring nodes, and the measurement of calculation time is not so appropriate. However, both before and after results are dealing exactly in the same way except with and without repeated heap value, then the result is fair to be used to determine the success of the method.

Speed up factor and calculation time

Figure 4 shows the calculation time of our final result. The calculation times are decreasing with increasing number of computational cores. However, the calculation time is increasing when amount of core is greater than 12.

Figure 5 shows the speed up of our code. The speed up is defined as

$$\text{speed up} = \frac{T_N^p}{T_1^p} \tag{4}$$

where T is calculation time, p indicate that the code is parallel code, and the subscript denote amount of operating cores.

From the result, we can see that the speed up is better when there is high amount of data, and the speed up dropping with increasing amout of cores.

DISCUSSION

Calculation time

Figure 6 shows the result of our code and from Yang and Stern (2017) is compared. The resolution of the figure is low but it is needed to make figures can be compared easily. Note

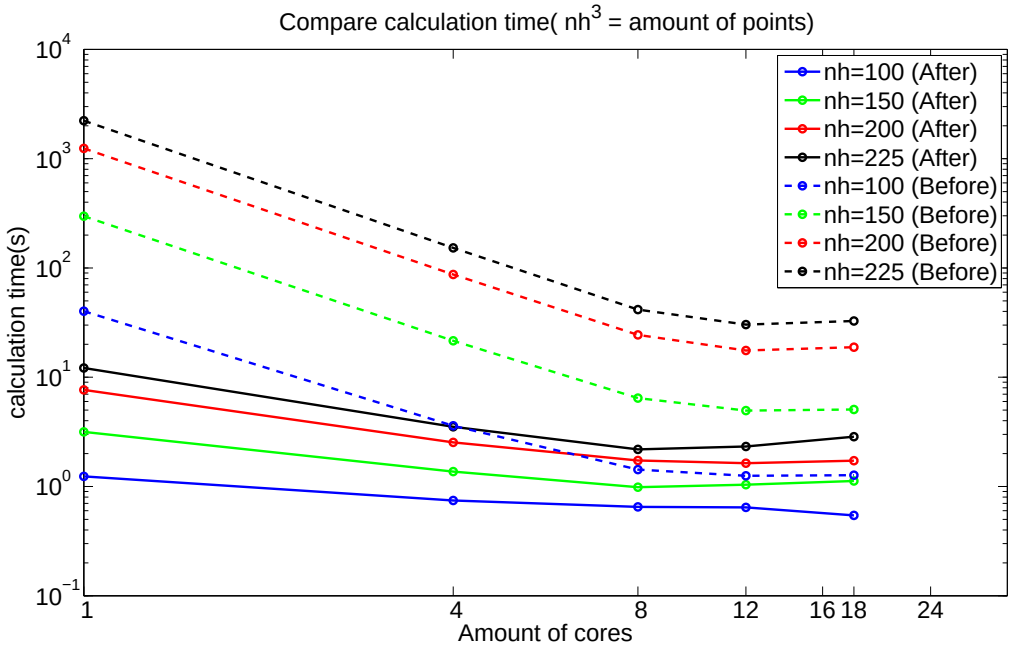

Figure 3: The comparison between before and after allowing heap to have repeated element. nh^3 is amount of points(old version of the code).

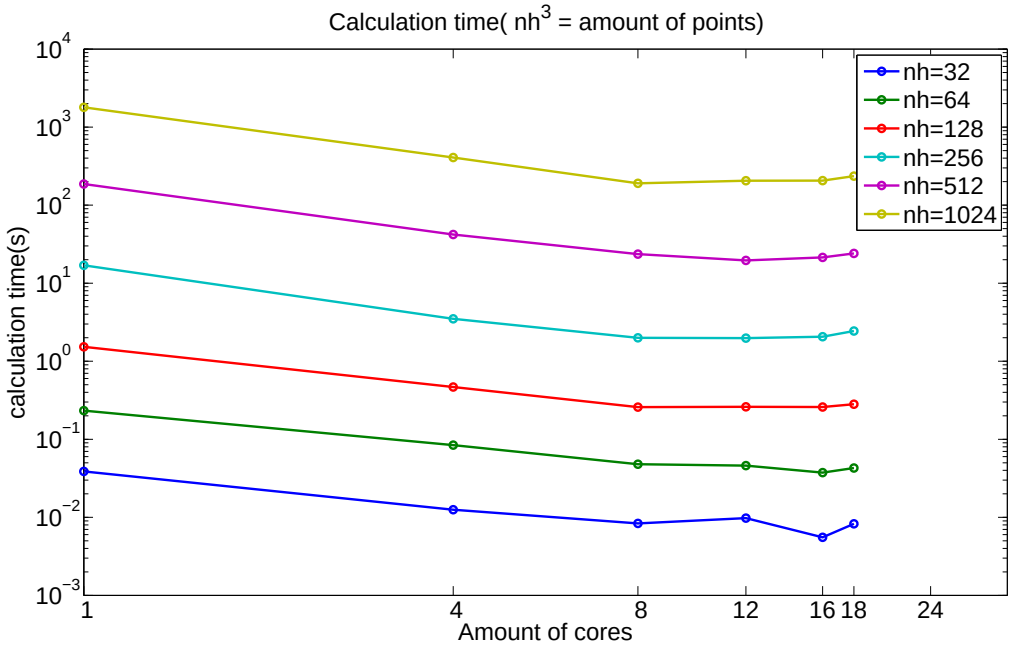

Figure 4: Calculation time of the final code(with selective synchronization and proper time measurement).

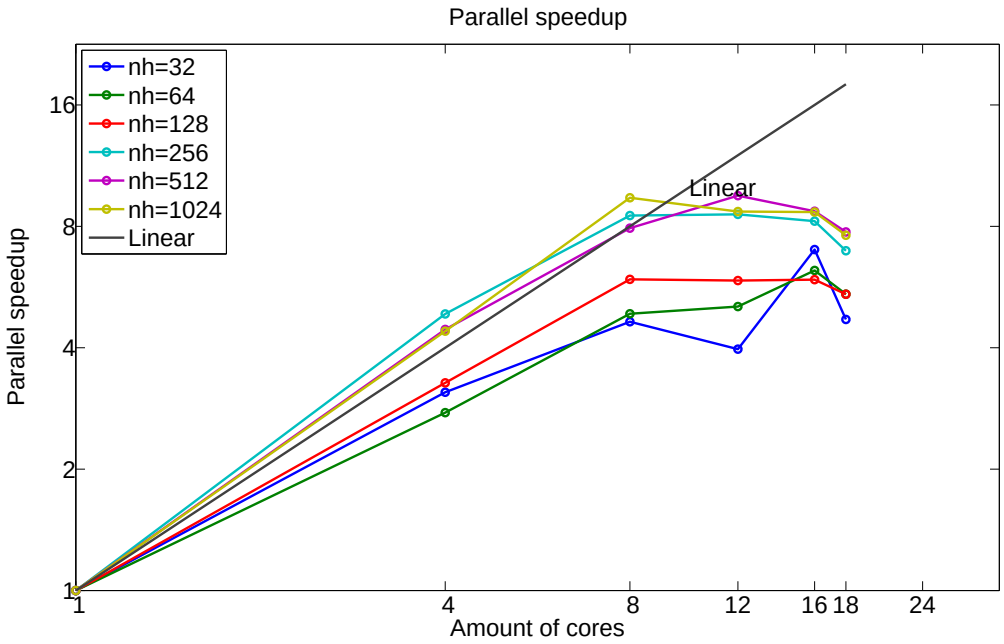

Figure 5: Speed up result.

also that the figure from Yang and Stern (2017) contain many value of stride, but in our work we consider only the cas of stride(δs) = $2.0\Delta h$. The comparison shows that our code performs better that the original work in the showed region. However, our computational resources are limited and can not test our code to be able to compare with Yang and Stern (2017) thoroughly.

Parallel speed up

Figure 6 shows the result of parallel speedup from Yang and Stern (2017). The graph is plot separately for each value of nh. Note again that, we are considering the black line where $\delta s = 2.0\Delta h$. Comparing with Figure 5 , we can see that our result is out-perform since the parallel speed up of our result are drop faster than Yang and Stern (2017).. Moreover, our result start to perform worse when amount of cores reached 16, but for Yang and Stern (2017) this happen at far more numbeer of cores.

CONCLUSION

In this work, we try to implement the parallel fastmarching method from Yang and Stern (2017). The suggested method is to decompose a domain in to several subdomain and updated each domain semi-separately. The update have to be stop frequently and implement synchronization before update further. The results of our program is better when comparing computational time but worse for parallel speed up. However, since we have limited computational resources, we cannot compare thoroughly with the original work. We also need to solve our scaling problem.

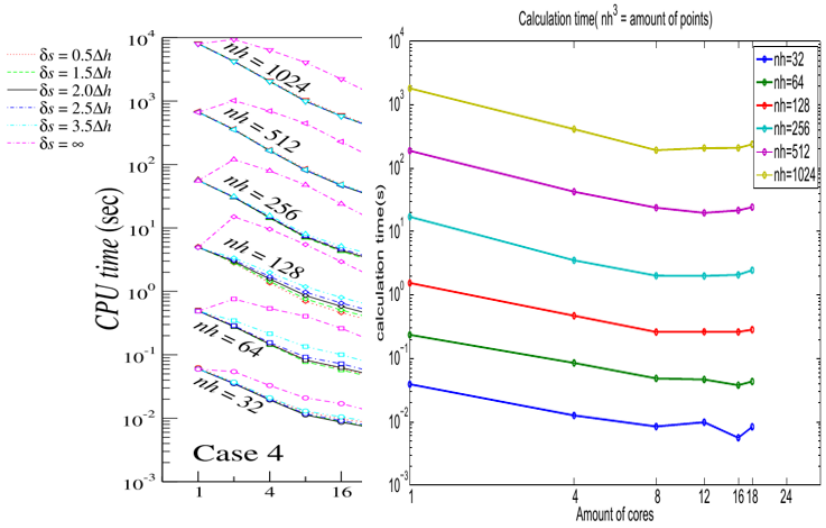

Figure 6: Comparing of calculation time between our result(right) and from Yang and Stern (2017)(left).

REFERENCES

Yang, J. and F. Stern, 2017, A highly scalable massively parallel fast marching method for the eikonal equation: Computational Physics, **332**, 333–362.

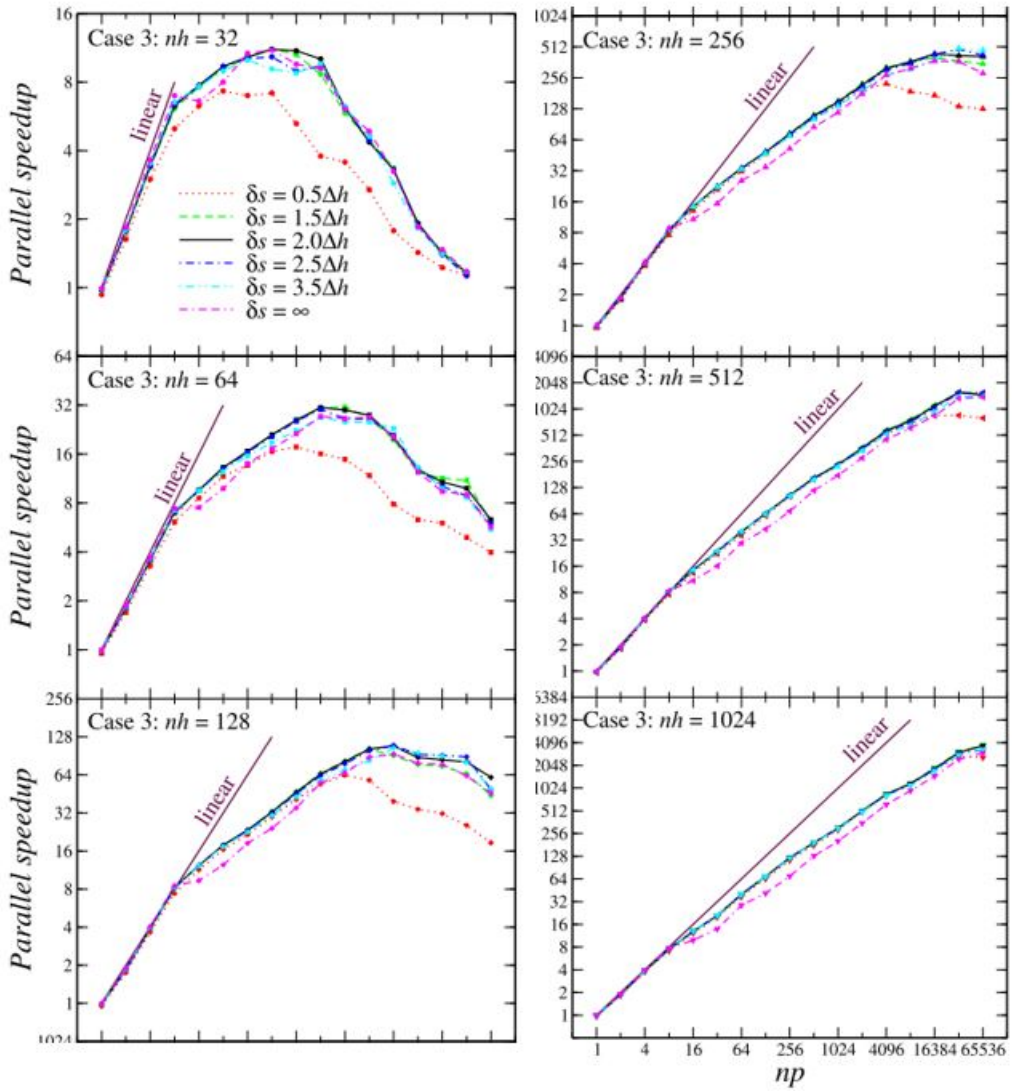

Figure 7: Parallel speed up result from Yang and Stern (2017)(left).