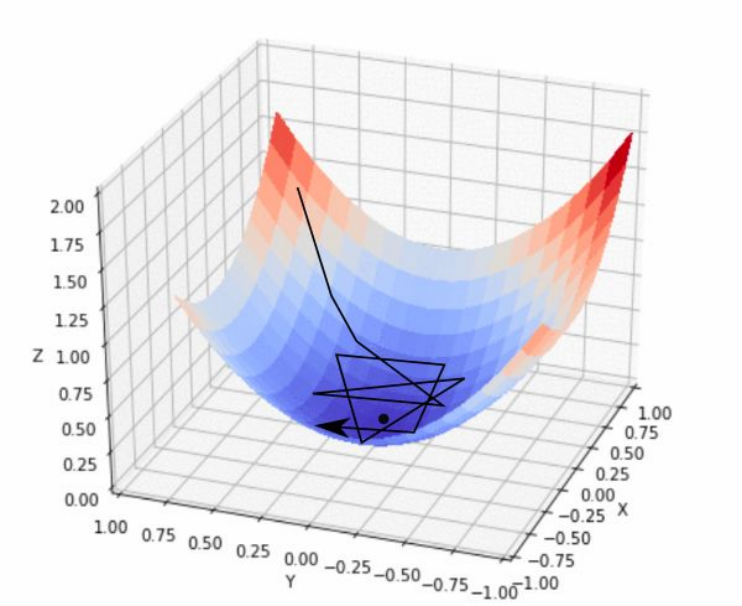
Parallel Stochastic Gradient Descent

Michael McKinsey, Phi Thuan Au, Yashas Salankimatt, Bora Oztekin

CSCE 435 - Parallel Computing

What is gradient descent?

Iterative numerical optimization process conducted by moving in the opposite direction of the gradient (direction of greatest change) until convergence



Sequential Algorithm

- Randomly initialize weights in linear equation

$$\hat{y} = w_1 x_1 + w_2 x_2 + w_n x_n + b$$

- Iteratively update weights in the opposite direction of the gradient **for each sample**

Algorithm 1: Gradient Descent

Result: Write here the result

Initialize at step $t = 0$ to $\mathbf{w}(0)$;

for $t = 0, 1, 2, \ldots$ **do**

  Compute the gradient;

  $\mathbf{g}_t = \nabla E_{in}(\mathbf{w}(t))$;

  $\mathbf{v}_t = -\mathbf{g}_t$;

  $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \mathbf{v}_t$;

end

`w(t+1) = w(t) + η * sample loss gradient`

Parallel Algorithm

- Randomly initialize weights in linear equation

$$\hat{y} = w_1 x_1 + w_2 x_2 + w_n x_n + b$$

- Iteratively update weights in the opposite direction of the gradient **for the entire epoch**

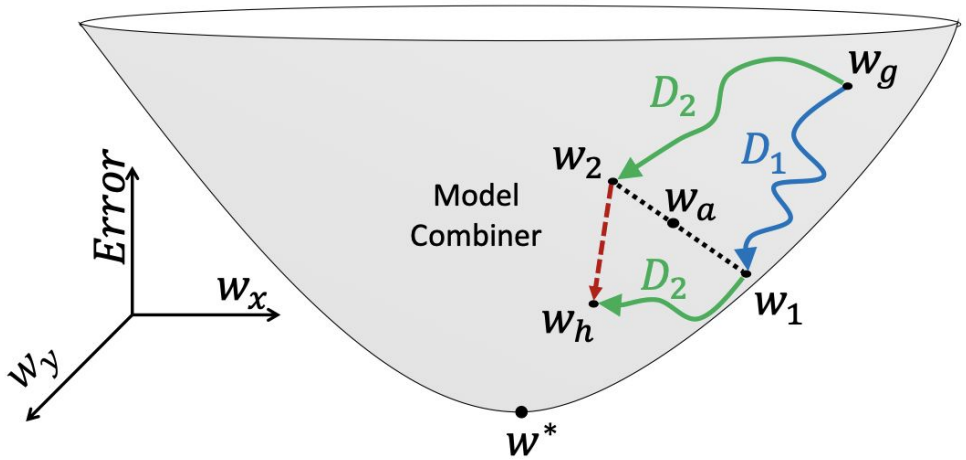
Algorithm 2: Parallel Gradient Descent

Result: Write here the result
Initialize at step $t = 0$ to $\mathbf{w}(0)$;
for $t = 0, 1, 2, \ldots$ **do**
  Each t here, unlike in sequential, has seen all samples;
  $\mathbf{v_t} = 0$;
  for $s = 0, 1, 2, \ldots$ **do**
    Compute the gradient for all samples in parallel;
    $\mathbf{g}_{ts} = \nabla E_{in}(\mathbf{w}(t))$;
    $\mathbf{v}_{ts} = -\mathbf{g}_t$;
    $\mathbf{v}_t = \mathbf{v}_t + \mathbf{v}_{ts}$;
  end
  $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \frac{\mathbf{v}_t}{|S|}$;
end

```
w(t+1) = w(t) + η * (grad 1 + grad 2 + grad 3 + …)
```

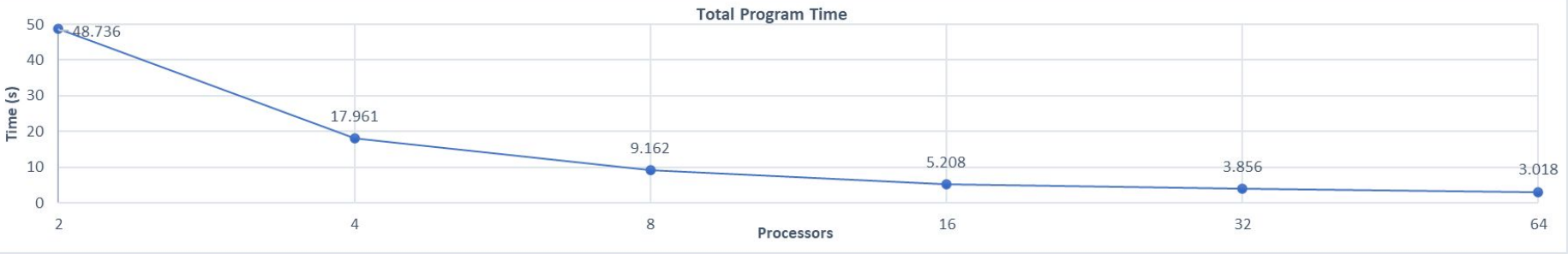
Parallel Algorithm

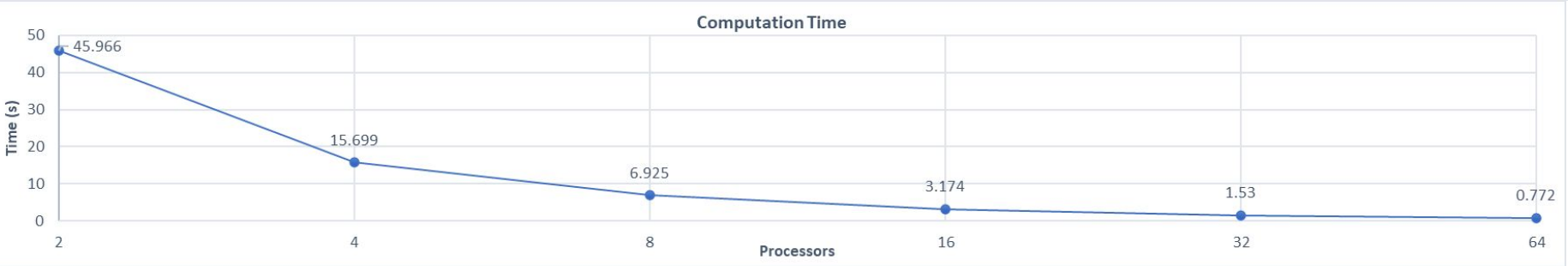
- Introduces permutation invariance since there is no randomness to the sequence used to update the weights
  - Can stabilize training process and possibly provide a speedup to convergence

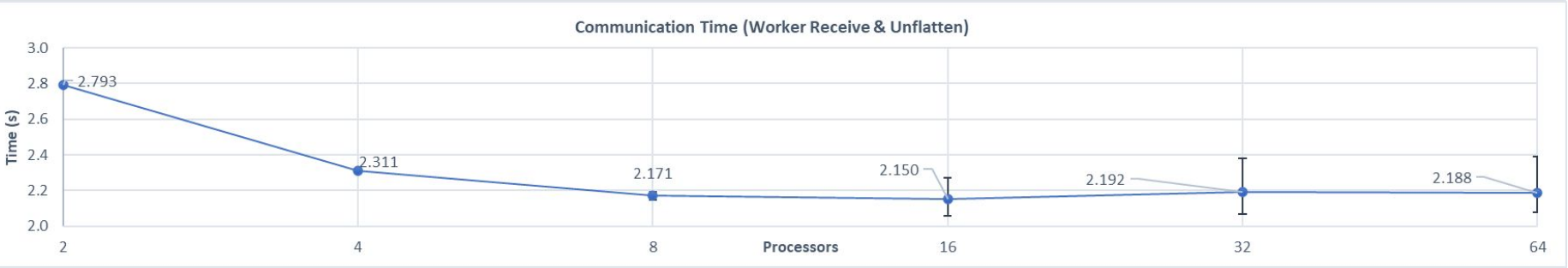


Message Passing Interface (MPI): Results (Strong Scaling)

(284k, 16 nodes, 4 cores/node, 8 GB/node, 0.01 LR, 15 Epochs)

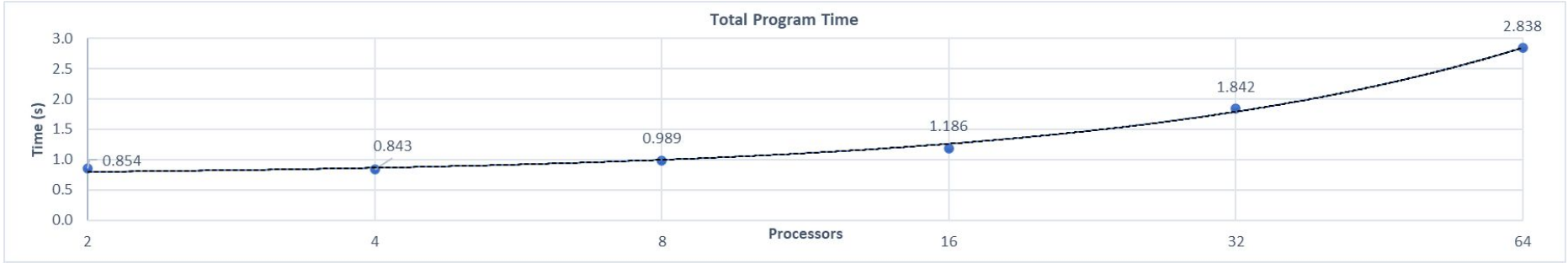


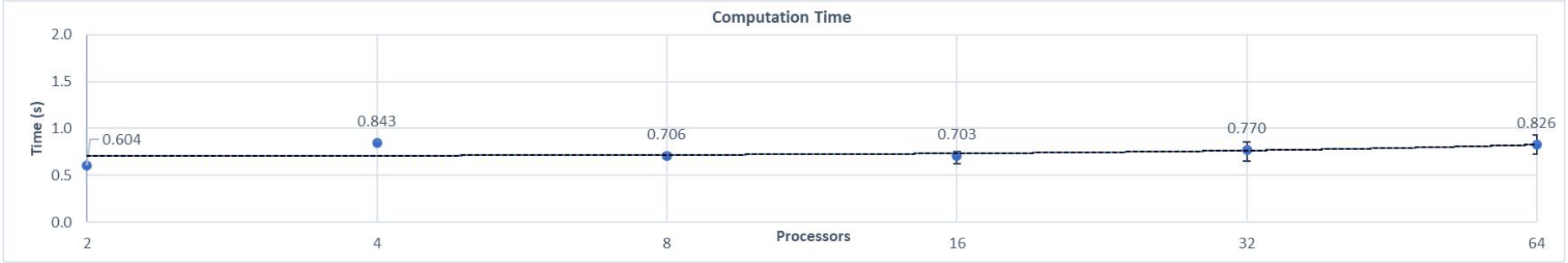


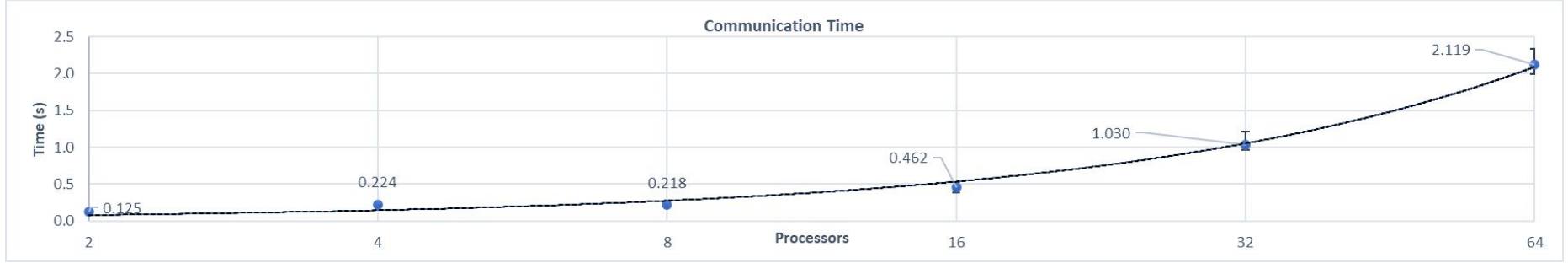


Message Passing Interface (MPI): Results (Weak Scaling)

(~4.5k Each, 16 nodes, 4 cores/node, 8 GB/node, 0.01 LR, 15 Epochs)



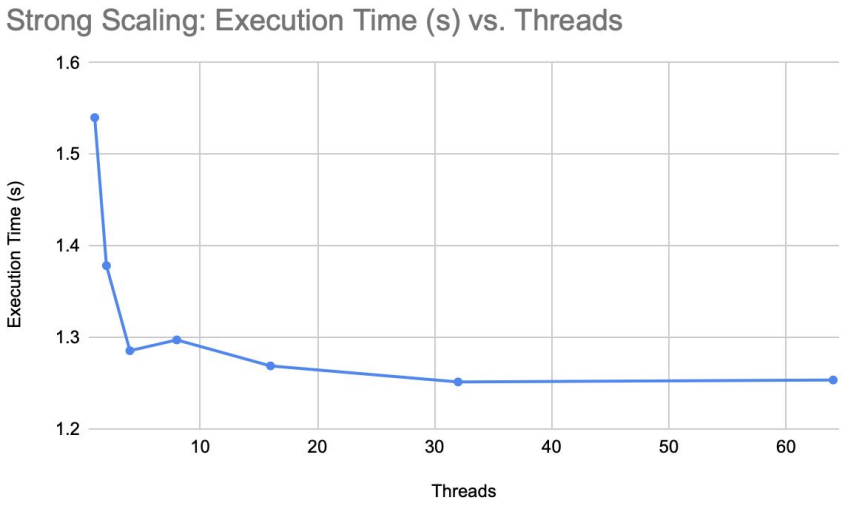


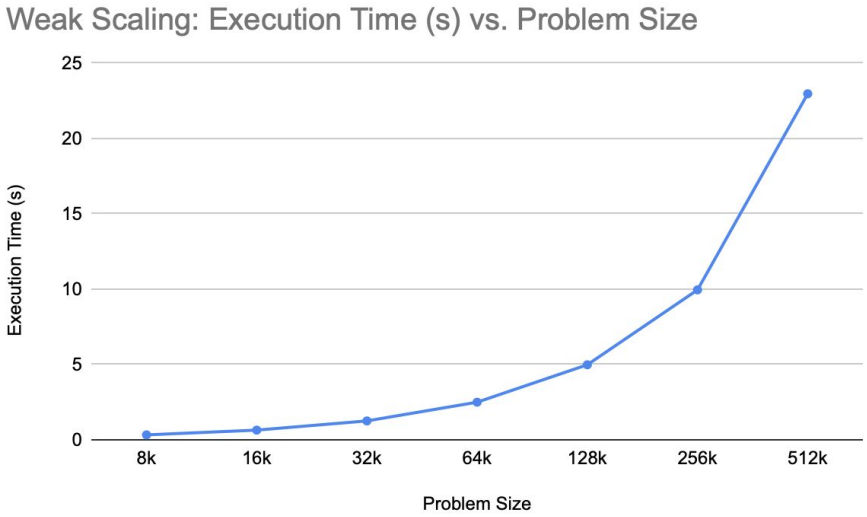


Message Passing Interface (MPI): Analysis & Findings

- Based on the empirical results of the experimentation with the MPI implementation, the parallel implementation of Stochastic Gradient Descent scaled with benefit up to 64 processes.
- While each increase in process count resulted in a reduction in total time, the overhead of interprocess communication yielded diminishing returns starting at 8 processes.

CUDA: Results





CUDA: Analysis & Findings

- Using a one-layer logistic regression mitigates the benefit from parallelism since there are not as many learnable parameters.
- With a very large dataset, different batching techniques may affect model performance. This may be an added benefit of using a GPU over a CPU.
- CUDA memory management for 2D arrays is quite difficult; therefore, we used the common practice of allocating large 1D arrays with i*N + j indexing.

Thank you!