CS205 Parallel Design of FaceX-Train

Zhecheng Yao, Yixian Gan, Rebecca Qiu, Lucy Li

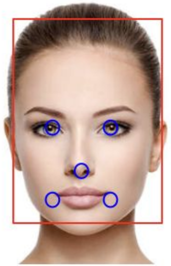
Introduction

FaceX-Train

1

3

2



- **Training Data**: consists of facial images and corresponding annotated facial landmarks.
- **Training Parameters**: specified in a configuration file

Challenges:

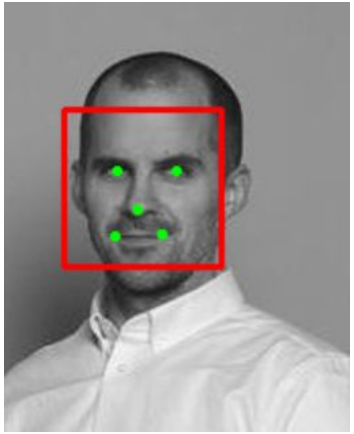
1. High data volume
   - The simplest training set consists of ~6000 training images, 260,000 rows
   - Transformation is done at the pixel by pixel level
2. Varying offsets
   - Each process needs to work on images of different dimensions
3. Storage
   - Facial rectangles
   - Landmark coordinates

Working Sequential Baseline Model

Training Time: ~ 5700s on cluster (Intel Xeon E5-2683 v4)
Data Size: 13466 images (250x250 pixel)

```
Training begin.
Training data count: 13466
(^_^) Finish training 1 regressor. Using 553.477s. 10 in total.
(^_^) Finish training 2 regressor. Using 561.787s. 10 in total.
(^_^) Finish training 3 regressor. Using 561.36s. 10 in total.
(^_^) Finish training 4 regressor. Using 564.956s. 10 in total.
(^_^) Finish training 5 regressor. Using 564.419s. 10 in total.
(^_^) Finish training 6 regressor. Using 568.158s. 10 in total.
(^_^) Finish training 7 regressor. Using 588.351s. 10 in total.
(^_^) Finish training 8 regressor. Using 578.051s. 10 in total.
(^_^) Finish training 9 regressor. Using 579.996s. 10 in total.
(^_^) Finish training 10 regressor. Using 575.504s. 10 in total.
```

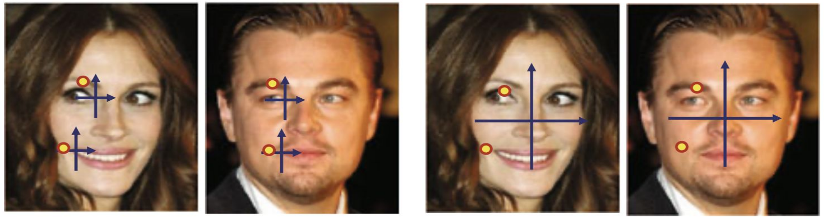
Training



Testing

Training: Two-level Boosted Regression

Training Size: 13466 Images



(a) (b)

Outer Regression

- Explicit Shape Regression Framework
- Stage regressors (R^1 , ..., R^T) are sequentially learnt to reduce the alignment error between the input shape S^T and the mean shape of all training images

$$R^t = \operatorname*{argmin}_R \sum_{i=1}^{N} ||y_i - R(I_i, S_i^{t-1})||_2$$

$$y_i = M_{S_i^{t-1}} \circ (\hat{S}_i - S_i^{t-1}),$$

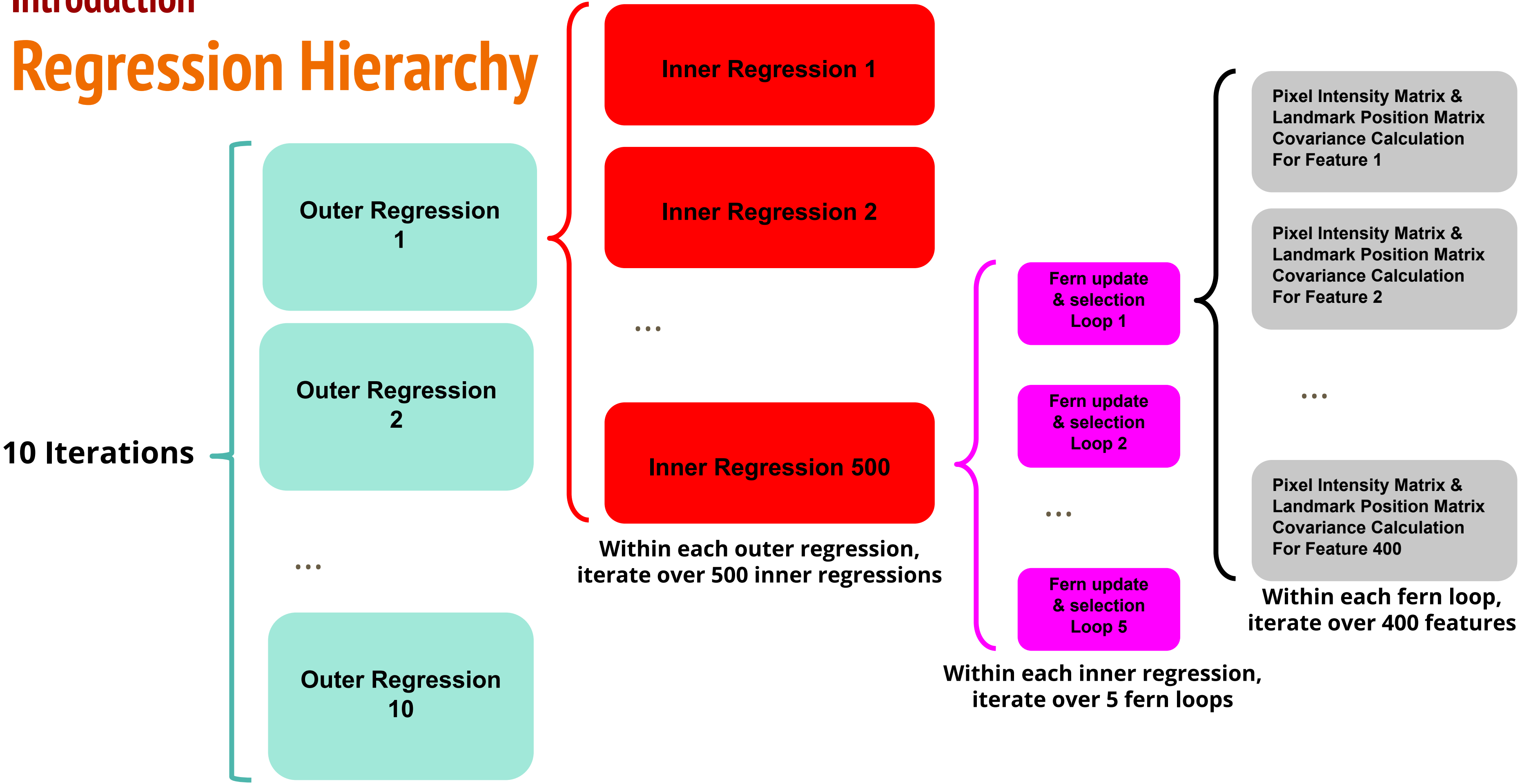
Inner Regression

- Ferns Framework
- Sequentially learnt to greedily fit regression targets
- Sample useful features distributed around salient landmarks, where feature that maximizes correlation with the target is selected

$$j_{\text{opt}} = \operatorname*{argmin}_j \ \text{corr}(Y_{\text{prob}}, X_j)$$

Introduction

Regression Hierarchy



Parallelization Overview

★ Non-trivial communication   Bottleneck

Algorithm 1 ESR Training

Require: Image I, Shape S

Ensure: Regressor R

    Read in data
    Augment training data
    for t in 1 to T **do**
        Compute similarity transform
        Compute Normalized Target
        $R_t \leftarrow$ LearnStageRegressor
        for i in 1 to N **do**
            Update landmark shape using Gradient Boost
        end for
    end for
    return Regressor R

Outer Regression

Algorithm 2 LearnStageRegressor

Require: Targets Y

Ensure: Stage Regressor R_t

    Generate local coordinates
    Extract shape indexed pixels
    Pre-compute pixel-pixel covariance
    $Y_0 \leftarrow$ Random initialization
    for k from 1 to K **do**
        Correlation based feature selection
        Sample F thresholds from an uniform distribution
        Partition training samples into 2^F bins
        Compute the outputs of all bins
        Construct a fern
        Update the targets
    end for
    $R_t \leftarrow$ Construct stage regressor
    return Stage regressor R_t

Fern Update Loop

& Feature Iterations

Inner Regression

Parallelization Overview

To Parallelize

★ Non-trivial communication   Bottleneck

Variables: Training images and labeled shapes $\{I_l, \hat{S}_l\}_{l=1}^{L}$; ESR model $\{R^t\}_{t=1}^{T}$; Testing image I; Predicted shape S; *TrainParams*{times of data augment N_{aug}, number of stages T};
TestParams{number of multiple initializations N_{int}};
InitSet which contains exemplar shapes for initialization

ESRTraining$(\{I_l, \hat{S}_l\}_{l=1}^{L}, \textit{TrainParams}, \textit{InitSet})$
// augment training data
$\{I_i, \hat{S}_i, S_i^0\}_{i=1}^{N} \leftarrow \textit{Initialization}\ (\{I_l, \hat{S}_l\}_{l=1}^{L}, N_{\text{aug}}, \textit{InitSet})$ 1
for t from 1 to T
★ $Y \leftarrow \{M_{S_i^{t-1}} \circ (\hat{S}_i - S_i^{t-1})\}_{i=1}^{N}$ // compute normalized targets 2
$R^t \leftarrow \textit{LearnStageRegressor}(Y, \{I_i, S_i^{t-1}\}_{i=1}^{N})$ // using Eq. (3)
for i from 1 to N
$S_i^t \leftarrow S_i^{t-1} + M_{S_i^{t-1}}^{-1} \circ R^t(I_i, S_i^{t-1})$
return $\{R^t\}_{t=1}^{T}$

★ $$M_S = \underset{M}{\operatorname{argmin}} \|\boxed{\bar{S}} - M \circ S\|_2,$$ 2

Outer regression

Variables: regression targets $Y \in \Re^{N \times 2N_{\text{fp}}}$; training images and corresponding estimated shapes $\{I_i, S_i\}_{i=1}^{N}$; training parameters *TrainParams*$\{N_{\text{fp}}, P, \kappa, F, K\}$; the stage regressor R; testing image and corresponding estimated shape $\{I, S\}$;

LearnStageRegressor$(Y, \{I_i, S_i\}_{i=1}^{N}, \textit{TrainParams})$
$\{\Delta_\alpha^{l_\alpha}\}_{\alpha=1}^{P} \leftarrow \textit{GenerateLocalCoordinates}(N_{\text{fp}}, P, \kappa)$
★ $\rho \leftarrow \textit{ExtractShapeIndexedPixels}(\{I_i, S_i\}_{i=1}^{N}, \{\Delta_\alpha^{l_\alpha}\}_{\alpha=1}^{P})$ 3
★ $\text{cov}(\rho) \leftarrow$ pre-compute pixel-pixel covariance 4
$Y^0 \leftarrow Y$ // initialization
for k from 1 to K
$\{\rho_{m_f} - \rho_{n_f}\}_{f=1}^{F}, \{m_f, n_f\}_{f=1}^{F} \leftarrow$
$\textit{CorrelationBasedFeatureSelection}(Y^{k-1}, \text{cov}(\rho), F)$ 5
$\{\theta_f\}_{f=1}^{F} \leftarrow$ sample F thresholds from an uniform distribution
$\{\Omega_b\}_{b=1}^{2^F} \leftarrow$ partition training samples into 2^F bins
$\{y_b\}_{b=1}^{2^F} \leftarrow$ compute the outputs of all bins using Eq. (7)
$r_k \leftarrow \{\{m_f, n_f\}_{f=1}^{F}, \{\theta_f\}_{f=1}^{F}, \{y_b\}_{b=1}^{2^F}\}$ // construct a fern
$Y^k \leftarrow Y^{k-1} - r^k(\{\rho_{m_f} - \rho_{n_f}\}_{f=1}^{F})$ // update the targets
★ $R \leftarrow \{\{r^k\}_{k=1}^{K}, \{\Delta_\alpha^{l_\alpha}\}_{\alpha=1}^{P}\}$ // construct stage regressor 6
return R

Inner regression

No Communication

Communication (gather & distribute)

Parallelization Flow Chart

1 Read in a subset of images and compute the mean shape of the subset.

2 Compute the mean shape of all subsets, augment the training data for more robust learning, and distribute to all ranks.

3 Compute **Normalization Target** (the normalized difference between ground-truth landmark points and the initial shapes of a training data point) and align the input shape to the mean shape using a similarity transformation, minimizing the L-2 distance.

4 Generate local coordinates for pixels in the training images.

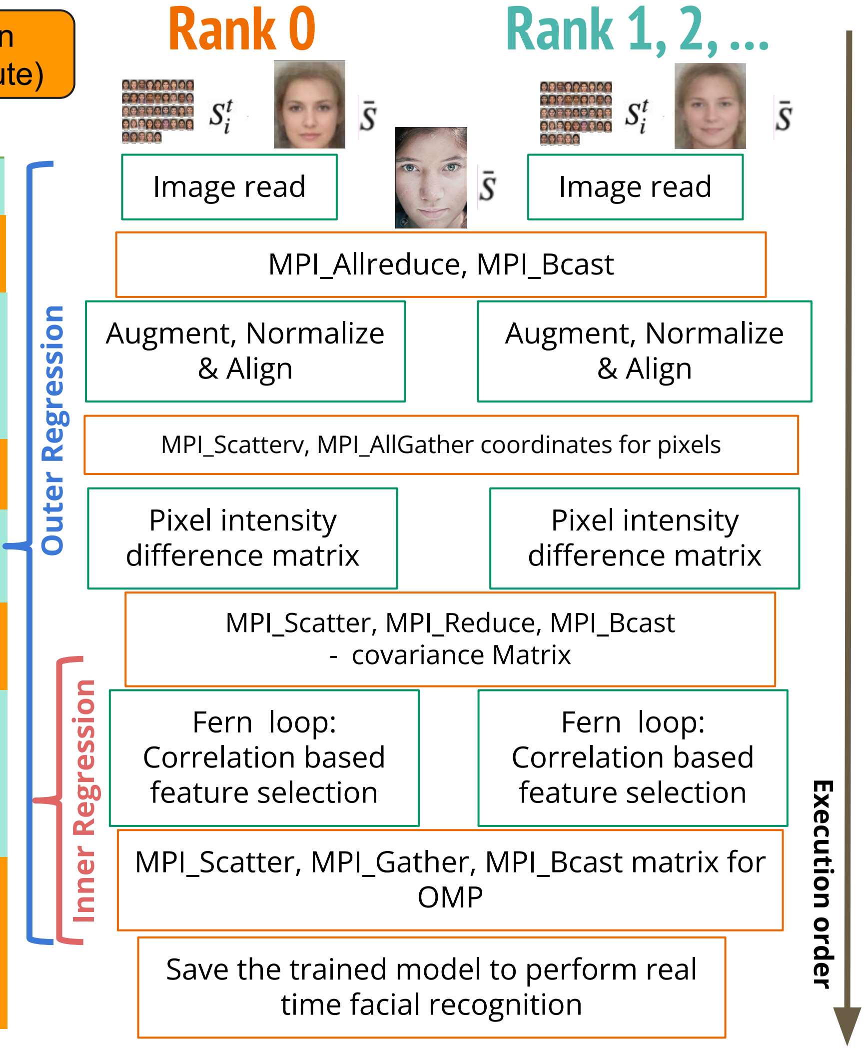
5 Extract the pixel-pair values at the randomly selected locations, and store the pixel values if the pixel position is inside the image bounds.

6 Compute the covariance matrix of the pixel intensity difference values and distribute to all ranks.

7 Train all Ferns in the inner regressor, which calculates and selects fern features by randomly projecting the input data onto a single dimension and then identifying the two pixels that have the strongest correlation with this projection.

8 Compress the inner regressor to reduce model size by calling the Orthogonal Matching Pursuit (OMP) function.

9 After the two-level boosted regression is completed, average the results from all ranks and save the trained model to file.

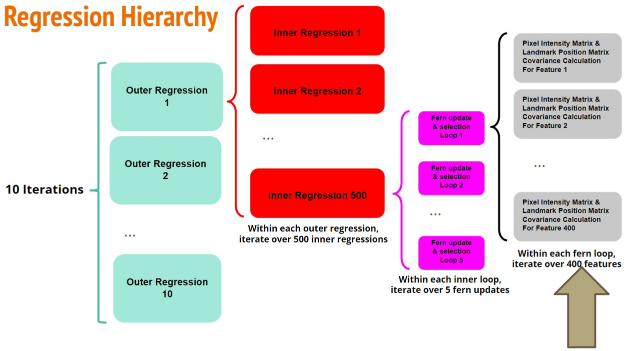


Identify Bottleneck Through Profiling

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
84.00   2462.49  2462.49 10025000     0.25     0.25  Covariance(double*, double*, int)
 3.30   2559.29    96.81     5000    19.36   517.39  FernTrain::Regress(std::vector<std::vector<cv:
 2.98   2646.77    87.48 1346600000     0.00     0.00  FernTrain::ApplyMini(cv::Mat, std::vector<do
 2.21   2711.55    64.78                             RegressorTrain::Apply(std::vector<cv::Point_<d
 1.99   2769.76    58.21                             cv::Mat::~Mat()
 1.46   2812.60    42.84 1346600000     0.00     0.00  FernTrain::Apply(cv::Mat) const
 0.95   2840.33    27.73 1349293200     0.00     0.00  ShapeAdjustment(std::vector<cv::Point_<doubl
 0.94   2867.89    27.55 1349293200     0.00     0.00  ShapeDifference(std::vector<cv::Point_<doubl
```

Top time consuming process Covariance

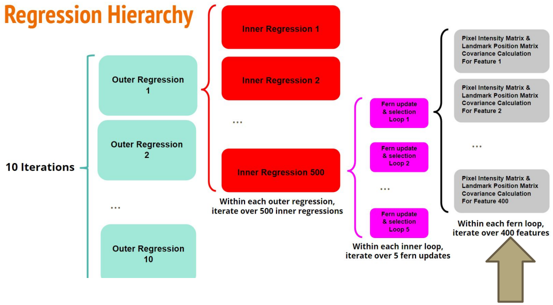
- Use majority of training time on avg (>80%)
- 10 million calls
- Called by each innermost iteration
- No further function calls (self seconds = cumulative seconds)



Covariance Kernel

```
double Covariance(double *x, double * y, const int size)
{
    double a = 0, b = 0, c = 0;
    for (int i = 0; i < size; ++i)
    {
        a += x[i];
        b += y[i];
        c += x[i] * y[i];
    }

    return c / size - (a / size) * (b / size);
}
```



Serial Baseline Performance:

- Size: 269320
- 4 flops per loop - 26930*4 flops per call
- function call 10^7 times
- 10772.8 GFlops
- 2462.49 s

= 4.37 GFlop/s

Operational Intensity:

- 4N operations
- 8*2N bytes

= 0.25 Flop/Byte

A closer look at our bottleneck

Roofline Analysis: Covariance Kernel

Intel Xeon E5-2683 16 core CPU

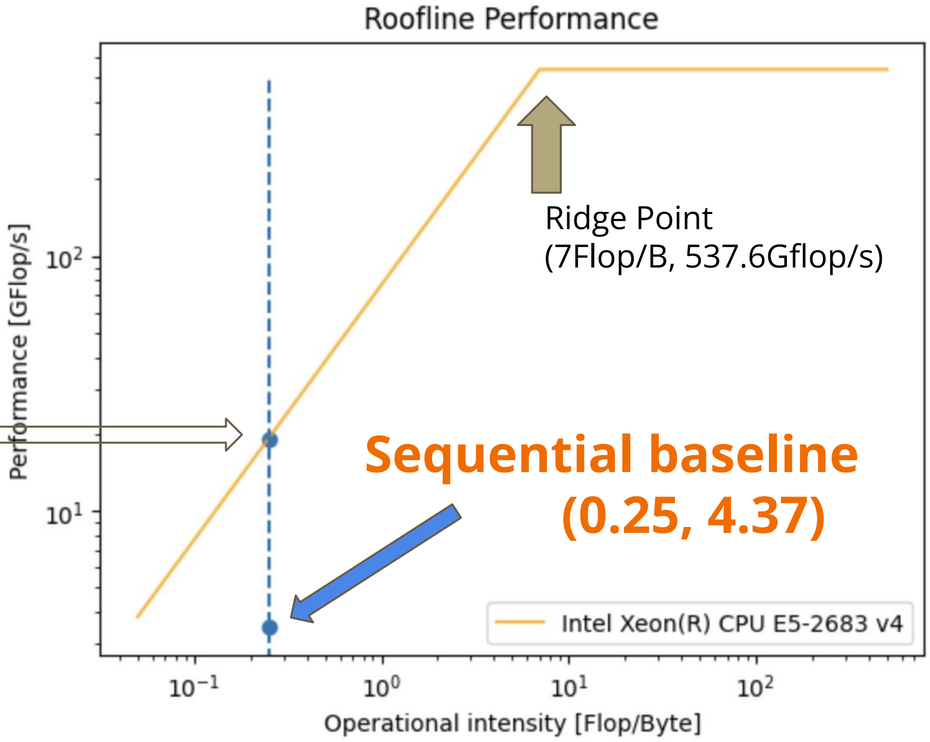
double precision peak π = 537.6Gflop/s

Memory bound up to 7 Flop/s

Peak memory bandwidth β = 76.8GB/s

Peak attainable performance

$$P = \min(\beta * I, \pi) = 19.2 \text{GFlop/s}$$



A closer look at our bottleneck

Parallelizing Covariance: Three Levels of Optimization

Distributed memory model: MPI

- Mainly collective operations
- MPI_Bcast, MPI_Scatter, MPI_Allgather, MPI_Allreduce, ...

For each process: Thread level parallelization

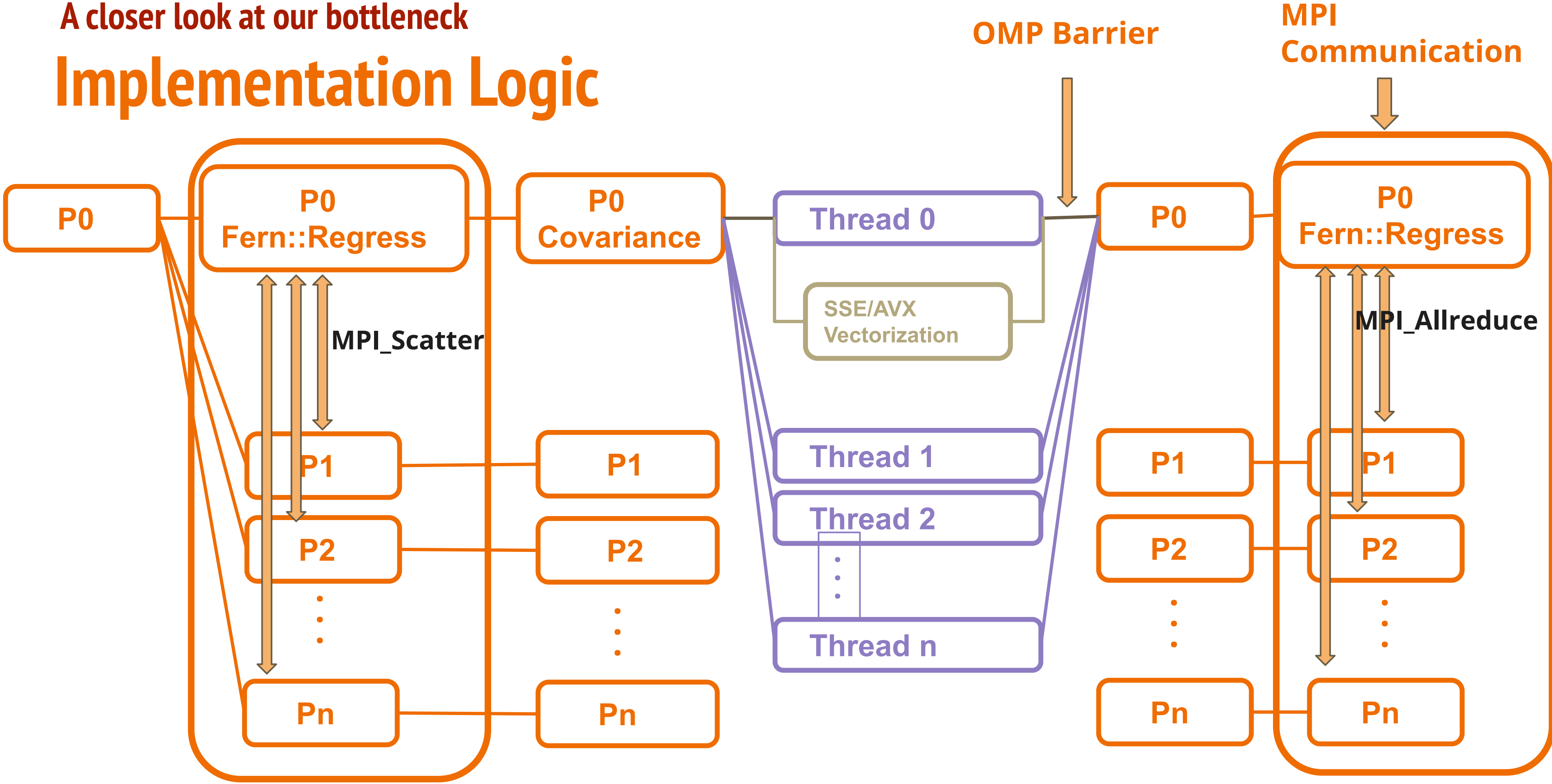
- OMP macros
- reduction

For each thread: Data level parallelization

- Vectorization
- SSE, AVX

A closer look at our bottleneck

Implementation Logic



Team 06

- Zhecheng Yao: zhechengyao@g.harvard.edu
- Yixian Gan: ygan@g.harvard.edu
- Rebecca Qiu: zqiu1@fas.harvard.edu
- Lucy Li: cli@fas.harvard.edu