CS205 Project Proposal
Team 06

Zhecheng Yao, Yixian Gan, Rebecca Qiu, Lucy Li

FaceX: Face Contour Recognition via Shape Regression

Face recognition

- Robustness: variations in face pose, expression, and lighting conditions
- Flexibility: detect a wide range of facial landmarks (e.g. eye, eyebrow, lips)
- Efficiency: relative lean and simple model to perform real time analysis

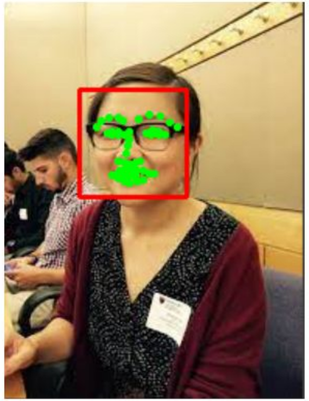
Our facial regression model aims to predict the position of facial landmarks, using a boosted regressor to make progressive inference, minimizing alignment error at each step.

Challenges:

- current version w/o parallelism has significant lagging in performance
- 1.5 seconds between face update and face recognition
- ~1 hour time requirement to train on a small dataset of 60 250 × 250 pixel image

Two Parts

- Face recognition w/ trained model
- Training model



Model and Data

Basic Pipeline of Recognition (FaceX):

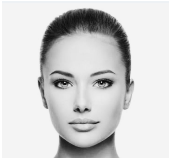
1. Load an image frame from camera, convert it to grayscale, maps onto a unit square and detect faces present.

2. For each detected face, draw a red rectangle to bound it.

3. align the initial facial landmarks to the face rectangle

Potential Optimizations

3. With OpenMP, split the input landmark into multiple batches and process each subset in parallel (reduction)



Model and Data

Basic Pipeline of Recognition (FaceX):

1. Load an image frame from camera, convert it to grayscale, maps onto a unit square and detect faces present.

2. For each detected face, draw a red rectangle to bound it.

3. Align the initial facial landmarks to the face rectangle

4. Apply trained regression model using pixel within the rectangle and initial facial landmarks as inputs to predict refined landmark locations.

5. Compute prediction medians as final landmarks' coordinates.

6. Draw green dots the final landmarks and display the aligned image.

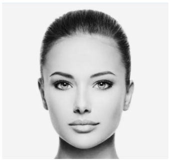
Data source:

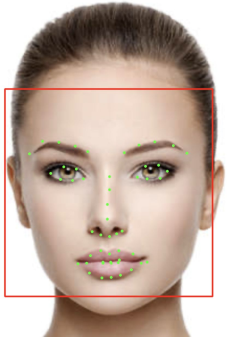
Images and videos taken on our phone or camera

Potential Optimizations

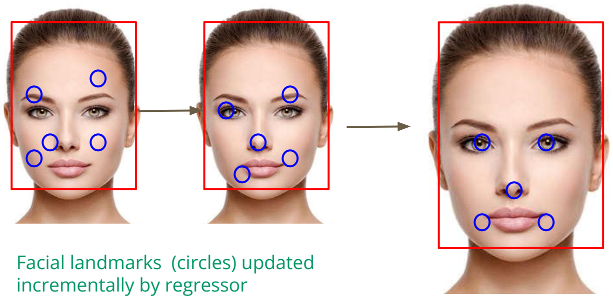
3. With OpenMP, split the input landmark into multiple batches and process each subset in parallel (reduction)

4. Use OpenMP to distribute the workload across multiple threads in the prediction stage





Training Model



Facial landmarks (circles) updated incrementally by regressor

FaceX-Train:

1. Creates a set of initial shapes for testing.

2. Maps the landmarks onto the unit square.

3. Augments Training data

4. Perform **gradient boosting regression -** two levels cascade, each iteration involves:

{

Randomly sampling P pixels, computing P^2 "**pixel-difference features**"

Features are indexed based on how far they are from local landmark

★ Feature selection based on **correlations - select F of the P^2 features**

Normalize regression target, aligning the target shape to the mean shape across all training dataset

★ Perform model compression using **orthogonal matching pursuit (OMP)**:

}

5. Returns the training parameters, training data, and test initial shapes.

★ Indicates Bottleneck in Sequential Version of the Code

{ Every step enclosed in brackets is individually parallelizable }

Potential Optimizations

★ In computing covariance, use vectorization (AVX 2 instructions), or OpenMP to parallelize the implementation

★ Using MPI, distribute the input shape across multiple processes

Profiling and Bottleneck

24.04 min	93.0%	0 s	⌄ main facex-train
23.95 min	92.7%	0 s	⌄ TrainModel(std::__1::vector<DataPoint, std::__1::allocator<DataPoint> > const&, TrainingParameters const&) facex-train
19.99 min	77.4%	0 s	⌄ RegressorTrain::Regress(std::__1::vector<cv::Point_<double>, std::__1::allocator<cv::Point_<double> > > const&, std::__1::vector<std::_
17.02 min	65.9%	0 s	⌄ FernTrain::Regress(std::__1::vector<std::__1::vector<cv::Point_<double>, std::__1::allocator<cv::Point_<double> > >, std::__1::allocat
16.68 min	64.5%	16.68 min	Covariance(double*, double*, int) facex-train
9.83 s	0.6%	8.95 s	› FernTrain::Regress(std::__1::vector<std::__1::vector<cv::Point_<double>, std::__1::allocator<cv::Point_<double> > >, std::__1::alloca
6.99 s	0.4%	0 s	› void* std::__1::__libcpp_operator_new<unsigned long>(unsigned long) [inlined] facex-train
2.64 s	0.1%	0 s	› cv::Point_<double>::Point_() [inlined] facex-train
736.00 ms	0.0%	4.00 ms	› cv::Mat::copyTo(cv::_OutputArray const&) const libopencv_core.4.7.0.dylib
256.00 ms	0.0%	256.00 ms	ShapeAdjustment(std::__1::vector<cv::Point_<double>, std::__1::allocator<cv::Point_<double> > > const&, std::__1::vector<cv::Point
166.00 ms	0.0%	6.00 ms	› cv::Mat::~Mat() libopencv_core.4.7.0.dylib
72.00 ms	0.0%	72.00 ms	cv::Mat::total() const libopencv_core.4.7.0.dylib
34.00 ms	0.0%	0 s	› cv::Mat::Mat(int, int, int) libopencv_core.4.7.0.dylib
34.00 ms	0.0%	34.00 ms	DYLD-STUB$$cv::Mat::total() const facex-train
18.00 ms	0.0%	0 s	› cv::RNG::fill(cv::_InputOutputArray const&, int, cv::_InputArray const&, cv::_InputArray const&, bool) libopencv_core.4.7.0.dylib
12.00 ms	0.0%	0 s	› std::__1::vector<double, std::__1::allocator<double> >::~vector() [inlined] facex-train
12.00 ms	0.0%	0 s	› cv::TLSDataContainer::getData() const libopencv_core.4.7.0.dylib
10.00 ms	0.0%	0 s	› operator new(unsigned long) libc++abi.dylib
10.00 ms	0.0%	0 s	› std::__1::vector<double, std::__1::allocator<double> >::vector(unsigned long) [inlined] facex-train
10.00 ms	0.0%	0 s	› std::__1::vector<std::__1::vector<cv::Point_<double>, std::__1::allocator<cv::Point_<double> > >, std::__1::allocator<std::__1::vecto
6.00 ms	0.0%	0 s	› void* std::__1::__libcpp_operator_new<unsigned long>(unsigned long) [inlined] facex-train
4.00 ms	0.0%	4.00 ms	cv::Mat::~Mat() libopencv_core.4.7.0.dylib
4.00 ms	0.0%	0 s	› void* std::__1::__libcpp_operator_new<unsigned long>(unsigned long) [inlined] facex-train
4.00 ms	0.0%	0 s	› void* std::__1::__libcpp_operator_new<unsigned long>(unsigned long) [inlined] facex-train
2.00 ms	0.0%	0 s	› cv::operator*(cv::Mat const&, cv::Mat const&) libopencv_core.4.7.0.dylib
2.00 ms	0.0%	2.00 ms	operator new[](unsigned long) libc++abi.dylib
2.00 ms	0.0%	2.00 ms	cv::theRNG() libopencv_core.4.7.0.dylib
1.79 min	6.9%	0 s	› RegressorTrain::CompressFerns() facex-train
15.06 s	0.9%	0 s	› cv::calcCovarMatrix(cv::_InputArray const&, cv::_OutputArray const&, cv::_InputOutputArray const&, int, int) libopencv_core.4.7.0.dy
9.71 s	0.6%	0 s	› void std::__1::__libcpp_operator_delete<void*>(void*) [inlined] facex-train
9.65 s	0.6%	0 s	› void* std::__1::__libcpp_operator_new<unsigned long>(unsigned long) [inlined] facex-train
9.62 s	0.6%	0 s	› cv::MatExpr::operator cv::Mat() const [inlined] facex-train
7.44 s	0.4%	0 s	› void* std::__1::__libcpp_operator_new<unsigned long>(unsigned long) [inlined] facex-train
7.28 s	0.4%	1.92 s	⌄ RegressorTrain::Regress(std::__1::vector<cv::Point_<double>, std::__1::allocator<cv::Point_<double> > > const&, std::__1::vector<std
3.90 s	0.2%	3.25 s	› FernTrain::Apply(cv::Mat) const facex-train
846.00 ms	0.0%	572.00 ms	› ShapeDifference(std::__1::vector<cv::Point_<double>, std::__1::allocator<cv::Point_<double> > > const&, std::__1::vector<cv::Point_
254.00 ms	0.0%	0 s	› cv::Point_<int>::inside(cv::Rect_<int> const&) const [inlined] facex-train
184.00 ms	0.0%	0 s	› std::__1::vector<cv::Point_<double>, std::__1::allocator<cv::Point_<double> > >::operator=(std::__1::vector<cv::Point_<double>, std

Profiling and Bottleneck

```
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);
}
```

Current performance:

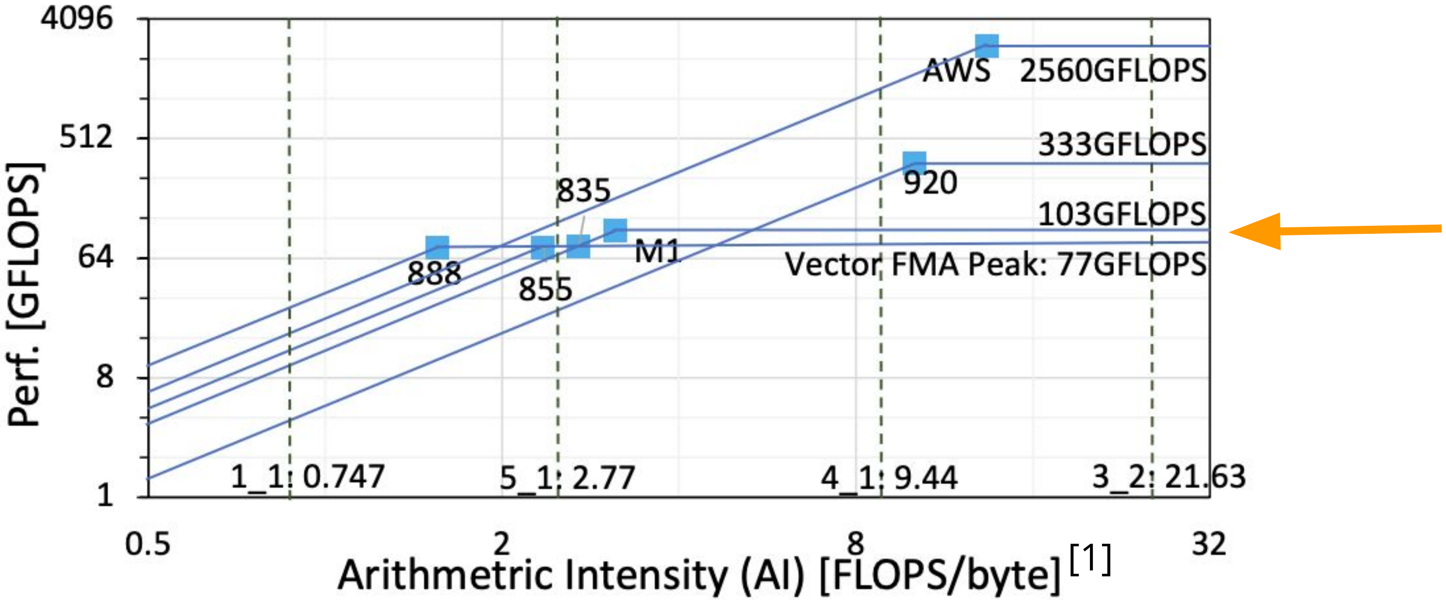
- size=13466 (# of training sample)
- 4*size per function call
- 400 function call per regressor
- 1000 regressor in total

= 21.54 GFlop

Time ≈ 1000s

0.02 GFlop/s

Roofline Analysis



[1]

[1]https://www.osti.gov/pages/servlets/purl/1863284

Parallelism Offered by the Project

Shared Memory Parallelism (OpenMP)

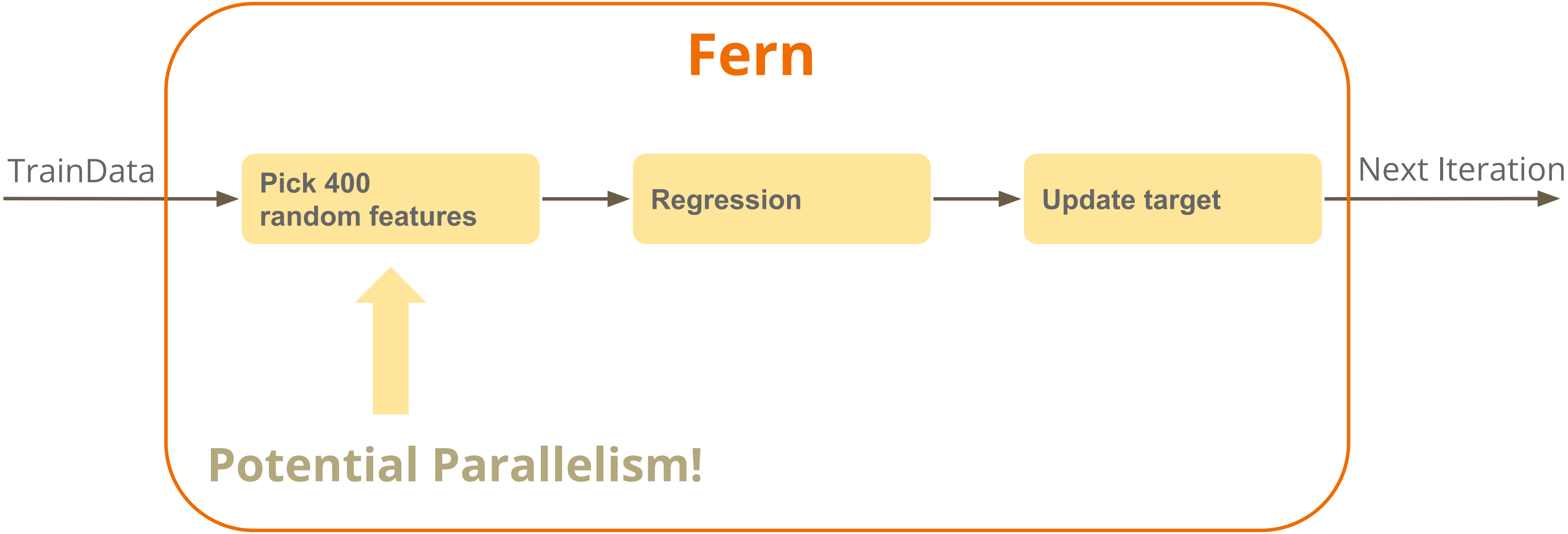
There are lots of For-loops to be parallelize

- utils_train.cpp::Covariance()
- regressor_train.cpp::Regress()
- regressor_train.cpp::CompressFerns()
- ...

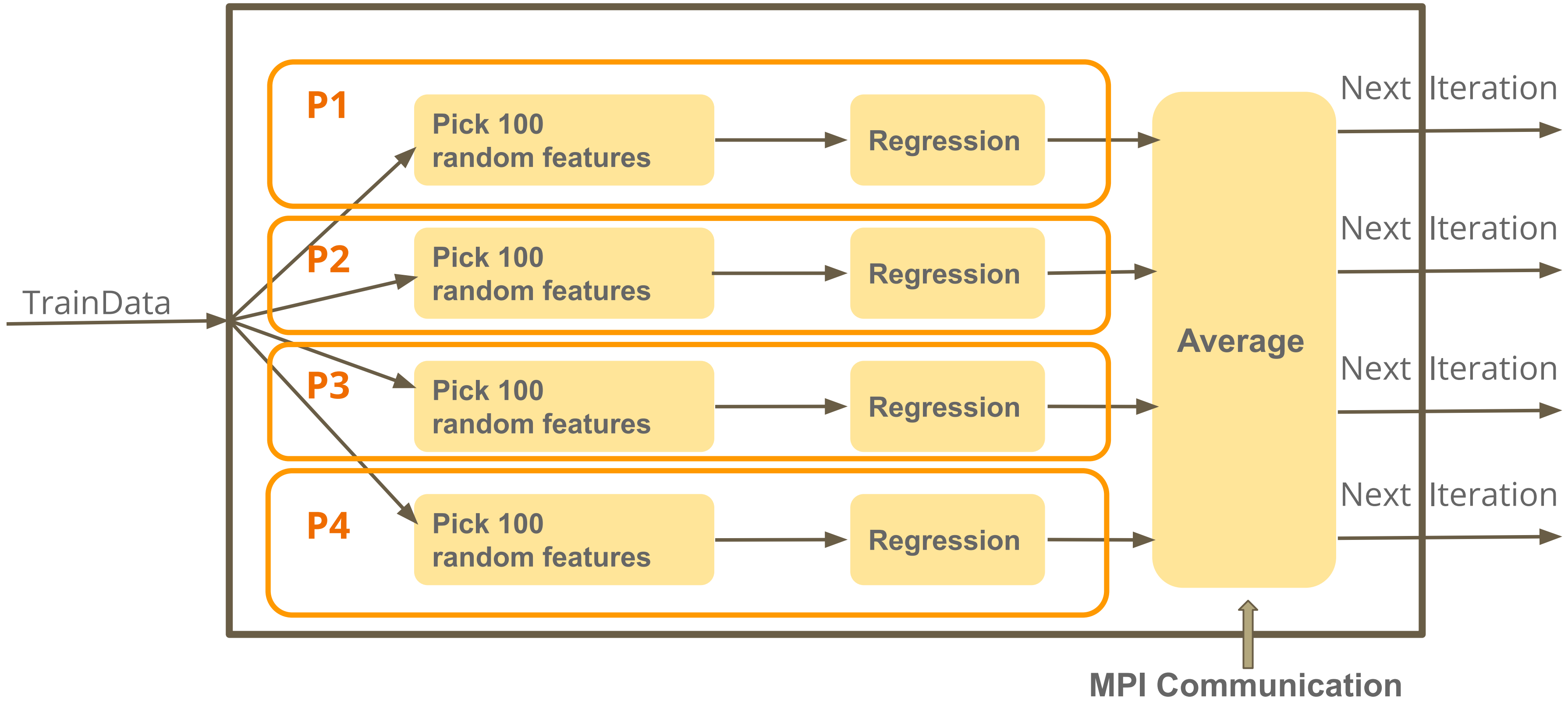
```
for (int i = 0; i < pixels_val.cols; ++i)
{
    Transform t = Procrustes(training_data[i].init_shape, mean_shape);
    vector<cv::Point2d> offsets(training_parameters_.P);
    for (int j = 0; j < training_parameters_.P; ++j)
        offsets[j] = pixels_[j].second;
    t.Apply(&offsets, false);

    for (int j = 0; j < training_parameters_.P; ++j)
    {

        cv::Point pixel_pos = training_data[i].init_shape[pixels_[j].first]
            + offsets[j];
        if (pixel_pos.inside(cv::Rect(0, 0,
            training_data[i].image.cols, training_data[i].image.rows)))
        {
            pixels_val.at<double>(j, i) =
                training_data[i].image.at<uchar>(pixel_pos);
        }
        else
            pixels_val.at<double>(j, i) = 0;
    }
}
```

Parallelism Offered by the Project

Distributed Memory Parallelism (MPI)



Parallelism Offered by the Project



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