DAFTAR PUSTAKA

Adreeva, O. (n.d.). *olya-d/eigenface: Implementation of the Eigenface algorithm for face recognition in C++*. Retrieved July 22, 2021, from https://github.com/olya-d/eigenface

Banana Pi BPI-M3 - Banana Pi Wiki. (n.d.). Retrieved December 29, 2020, from http://wiki.banana-pi.org/Banana_Pi_BPI-M3

Borkar, N. R., & Kuwelkar, S. (2018). Real-Time implementation of face recognition system. *Proceedings of the International Conference on Computing Methodologies and Communication, ICCMC 2017*, *2018-Janua*(Iccmc), 249–255. https://doi.org/10.1109/ICCMC.2017.8282685

Che, H., & Nguyen, M. (2014). Amdahl's law for multithreaded multicore processors. *Journal of Parallel and Distributed Computing*, *74*(10), 3056–3069. https://doi.org/10.1016/j.jpdc.2014.06.012

Govindaraj, V. (2016). *Parallel Programming in Raspberry Pi Cluster*.

Gustafson, J. L. (1988). Reevaluating amdahl's law. *Communications of the ACM*, *31*(5), 532–533. https://doi.org/10.1145/42411.42415

Home - OpenMP. (n.d.). Retrieved November 25, 2020, from https://www.openmp.org/

Ikram, A. (n.d.). *ikram1110/facereco-eigenface*. Retrieved July 22, 2021, from https://github.com/ikram1110/facereco-eigenface

Kholod, I. I., Rodionov, S. V., Tarasov, K. A., & Malov, A. V. (2017). Using

the features of functional programming for parallel building of decision trees. *Proceedings of the 2017 IEEE Russia Section Young Researchers in Electrical and Electronic Engineering Conference, ElConRus 2017*, 445–449. https://doi.org/10.1109/EIConRus.2017.7910587

Kurniawan, V., Wicaksana, A., & Prasetiyowati, M. I. (2017). The implementation of eigenface algorithm for face recognition in attendance system. *Proceedings of 2017 4th International Conference on New Media Studies, CONMEDIA 2017*, *2018-Janua*, 118–124. https://doi.org/10.1109/CONMEDIA.2017.8266042

Li, Y., & Zhang, Z. (2019). Parallel Computing: Review and Perspective. *Proceedings - 2018 5th International Conference on Information Science and Control Engineering, ICISCE 2018*, 365–369. https://doi.org/10.1109/ICISCE.2018.00083

Lindner, T., Wyrwal, D., Bialek, M., & Nowak, P. (2020). Face recognition system based on a single-board computer. *15th International Conference Mechatronic Systems and Materials, MSM 2020*, 1–6. https://doi.org/10.1109/MSM49833.2020.9201668

Mathews, M., & Abraham, J. P. (2017). Automatic Code Parallelization with OpenMP task constructs. *Proceedings - 2016 International Conference on Information Science, ICIS 2016*, 233–238. https://doi.org/10.1109/INFOSCI.2016.7845333

Matthews, S. J., Adams, J. C., Brown, R. A., & Shoop, E. (2018). Portable

parallel computing with the raspberry Pi. *SIGCSE 2018 - Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, *2018-Janua*, 92–97. https://doi.org/10.1145/3159450.3159558

Mustafa, B., Shahana, R., & Ahmed, W. (2015). Parallel implementation of Doolittle Algorithm using OpenMP for multicore machines. *Souvenir of the 2015 IEEE International Advance Computing Conference, IACC 2015*, 575–578. https://doi.org/10.1109/IADCC.2015.7154772

Owais, M., Shaikh, A., Jalal, A. A., & Hassan, M. M. (2019). Human Face Recognition using PCA Eigenfaces. *ICETAS 2019 - 2019 6th IEEE International Conference on Engineering, Technologies and Applied Sciences*. https://doi.org/10.1109/ICETAS48360.2019.9117489

Pei, S., Kim, M. S., & Gaudiot, J. L. (2016). Extending Amdahl's Law for Heterogeneous Multicore Processor with Consideration of the Overhead of Data Preparation. *IEEE Embedded Systems Letters*, *8*(1), 26–29. https://doi.org/10.1109/LES.2016.2519521

Poskanzer, J. (1989). *PGM Format Specification*. http://netpbm.sourceforge.net/doc/pgm.html

Putranto, E. B., Situmorang, P. A., & Girsang, A. S. (2017). Face recognition using eigenface with naive Bayes. *Proceedings - 11th 2016 International Conference on Knowledge, Information and Creativity Support Systems, KICSS 2016*, *4*, 9–12. https://doi.org/10.1109/KICSS.2016.7951418

Qu, X., Wei, T., Peng, C., & Du, P. (2018). A Fast Face Recognition System Based on Deep Learning. *Proceedings - 2018 11th International Symposium on Computational Intelligence and Design, ISCID 2018*, *1*, 289–292. https://doi.org/10.1109/ISCID.2018.00072

Ramadhani, A. L., Musa, P., & Wibowo, E. P. (2018). Human face recognition application using PCA and eigenface approach. *Proceedings of the 2nd International Conference on Informatics and Computing, ICIC 2017*, *2018-Janua*, 1–5. https://doi.org/10.1109/IAC.2017.8280652

Salehian, S., Liu, J., & Yan, Y. (2017). Comparison of threading programming models. *Proceedings - 2017 IEEE 31st International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2017*, 766–774. https://doi.org/10.1109/IPDPSW.2017.141

Sapna, S., Anjali, R., & Kamath, S. N. (2019). Performance Analysis of Parallel Implementation of PCA-based Face Recognition using OpenCL. *2019 4th IEEE International Conference on Recent Trends on Electronics, Information, Communication and Technology, RTEICT 2019 - Proceedings*, 877–881. https://doi.org/10.1109/RTEICT46194.2019.9016732

Suhendra, A., Wijaya, H., & Benny Mutiara, A. (2018). Analysis of comparison between sequential and parallel computation using openmp for Molecular Dynamic Simulation. *Proceedings of the 2nd International Conference on Informatics and Computing, ICIC 2017*,

2018-Janua, 1–5. https://doi.org/10.1109/IAC.2017.8280585

Tabassum, T., Charles, A., & Patil, A. V. (2016). Multicore versus Multiprocessor: A Review. *International Journal of Innovative Research in Computer and Communication Engineering (An ISO Certified Organization)*, *4*(1), 227–232. https://doi.org/10.15680/IJIRCCE.2016. 0401044

Wahyu Mulyono, I. U., Ignatius Moses Setiadi, D. R., Susanto, A., Rachmawanto, E. H., Fahmi, A., & Muljono. (2019). Performance Analysis of Face Recognition using Eigenface Approach. *Proceedings - 2019 International Seminar on Application for Technology of Information and Communication: Industry 4.0: Retrospect, Prospect, and Challenges, ISemantic 2019*, 12–16. https://doi.org/10.1109/ISEMANTIC.2019.8884225

Yang, C. T., Huang, C. L., Lin, C. F., & Chang, T. C. (2010). Hybrid parallel programming on GPU clusters. *Proceedings - International Symposium on Parallel and Distributed Processing with Applications, ISPA 2010*, 142–147. https://doi.org/10.1109/ISPA.2010.97

Zafaruddin, G. M., & Fadewar, H. S. (2018). Face recognition using eigenfaces. In *Advances in Intelligent Systems and Computing* (Vol. 810). Springer Singapore. https://doi.org/10.1007/978-981-13-1513-8_87

LAMPIRAN

LAMPIRAN I BARIS KODE PROGRAM

Baris Kode Program Serial

```
#include <iostream>
#include <iomanip>
#include <sstream>
#include <fstream>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <algorithm>
#include <omp.h>
#include <time.h>

#define MIN(a,b) (((a)<(b))?(a):(b))
const int Faces = 40;
const int Samples = 9;
const int Width = 92;    //  92 | 296 | 394 | 493
const int Height = 112;  // 112 | 360 | 480 | 600
int Eigenfaces = 28;
const std::string DataPath = "faces/";  // faces | face360 | face4
80 | face600
const int N = Faces;
const int M = Width * Height;
const std::string SampleName = "10";
const int MaxValue = 255;
double **facearray, **A, **B, **S, **V, **U, **W, **X, **P, **Atra
ns, **tempS;
double first, t, tsc;

void read_training_data() {
  facearray = (double**) malloc(Samples*sizeof(double*));
  for(int x=0; x<Samples; x++) {
    facearray[x] = (double*) malloc(M*sizeof(double));
    memset(facearray[x],0,M*sizeof(double));
  }
  for(int face=0; face<Faces; ++face) {
    // perulangan setiap foto
    for(int sample=0; sample<Samples; ++sample) {
      std::stringstream filename;
      filename << DataPath << "s" << face + 1 << "/" << sample  +
1 << ".pgm";
      // filename << DataPath << face + 1 << "_" << sample  + 1 <<
 ".pgm";
      std::ifstream image(filename.str().c_str());

      if (image.is_open()) {
        // baca foto

        std::string line;
        getline(image, line); // Skip P2 line
        getline(image, line); // Skip width line
        getline(image, line); // Skip height line
        getline(image, line); // Skip max value line
```

```
        int val;
        int x = 0;
        while(image >> val) {
        facearray[sample][x] = val;
          x++;
        }

        image.close();
      } else {
        std::cout << "Image was not opened.";
      }
    }

    // mencari citra rata-rata
    for(int x=0; x<M; ++x) {
      double sum = 0;
      for(int y=0; y<Samples; ++y) {
        sum += facearray[y][x];
      }
      A[face][x] = sum/Samples;
    }
  }
  free(facearray);
}

void create_mean_image() {
  // temukan mean image
  tsc = omp_get_wtime();
  for(int c=0; c<M; ++c) {
    double sum = 0;
    for(int r=0; r<N; ++r) {
      sum += A[r][c];
    }
    B[0][c] = sum / N;
  }
  tsc = omp_get_wtime() - tsc;
  printf("Execution time create mean image : %.2fms\n", tsc*1000);

  // output citra rata-rata
  std::stringstream filename;
  filename << "output/meanimage.pgm";
  std::ofstream image_file(filename.str().c_str());
  image_file << "P2" << std::endl << Width << std::endl << Height
<< std::endl << MaxValue << std::endl;
  for(int x=0; x<M; ++x) {
    int val = B[0][x];
    if(val < 0) {
      val = 0;
    }
    image_file << val << " ";
  }
  image_file.close();
}

void normalized() {
```

```
  // kurangkan mean dari setiap gambar
  tsc = omp_get_wtime();
  for(int r=0; r<N; ++r) {
    for(int c=0; c<M; ++c) {
      A[r][c] -= B[0][c];
      if(A[r][c] < 0) {
        A[r][c] = 0;
      }
    }
  }
  tsc = omp_get_wtime() - tsc;
  printf("Execution time mean subtraction : %.2fms\n", tsc*1000);

  // output citra normalisasi
  for(int x=0; x<N; ++x) {
    std::ostringstream filename;
    filename << "output/normalized/" << x << ".pgm";
    std::ofstream image_file(filename.str().c_str());
    image_file << "P2" << std::endl << Width << std::endl << Heigh
t << std::endl << MaxValue << std::endl;
    for(int y=0; y<M; ++y) {
      int val = A[x][y];
      if(val < 0) {
        val = 0;
      }
      image_file << val << " ";
    }
    image_file.close();
  }
}

void transpose_matrixA() {
  for(int r=0; r<M; ++r) {
    for(int c=0; c<N; ++c) {
      Atrans[r][c] = A[c][r];
    }
  }
}

void get_covariant_matrix() {
  for(int r=0; r<N; ++r) {
    for(int c=0; c<N; ++c) {
      S[r][c] = 0;
      for(int k=0; k<M; ++k) {
        S[r][c] += A[r][k] * Atrans[k][c];
      }
    }
  }
}

void calculate_eigenvalues() {
  // eigenvector pada matriks covarian
  // eigensystem(S).second.transpose()
  for(int r=0; r<N; ++r) {
    for(int c=0; c<N; ++c) {
      if (c == r) {
```

```
      P[c][r] = 1;
    }
  }
}
// std::cout<<"P[0][2] : "<<P[0][2]<<std::endl;
int max_rotation = 5 * N * N;
for(int i=0; i<N; ++i) {
  for(int j=0; j<N; ++j) {
    tempS[i][j] = S[i][j];
  }
}

double *eigenvalues;
eigenvalues = (double*) malloc(N*sizeof(double));
int small = 0;

for(int it=0; it<max_rotation; ++it) {
  double max = 0;
  int k, l;
  // temukan elemen off-diagonal terbesar
  for(int r=0; r<N-1; ++r) {
    for(int c=r+1; c<N; ++c) {
      if (fabs(tempS[r][c]) >= max) {
        max = fabs(tempS[r][c]);
        k = r;
        l = c;
      }
    }
  }

  if(max < 1.0e-12) {
    for(int i=0; i<N; ++i) {
      eigenvalues[i] = tempS[i][i];
    }
    // normalisasi P
    for(int c=0; c<N; ++c) {
      double length = 0;
      for(int r=0; r<N; ++r) {
        length += P[r][c] * P[r][c];
      }
      for(int r=0; r<N; ++r) {
        P[r][c] = P[r][c] / length;
      }
    }
    small = 1;
    break;
  }
  else {
    // melakukan rotasi
    double diff = tempS[l][l] - tempS[k][k];

    double t;
    if(fabs(tempS[k][l]) < fabs(diff)*1.0e-36) {
      t = tempS[k][l] / diff;
    }
    else {
```

```
      double phi = diff / (2.0 * tempS[k][l]);
      t = 1.0 / (fabs(phi) + sqrt(phi * phi + 1.0));
      if(phi < 0) {
        t = -t;
      }
    }
    double c = 1.0 / sqrt(t * t + 1.0);
    double s = t * c;
    double tau = s / (1.0 + c);
    double temp = tempS[k][l];
    tempS[k][l] = 0;
    tempS[k][k] = tempS[k][k] - t * temp;
    tempS[l][l] = tempS[l][l] + t * temp;

    for(int i=0; i<k; ++i) {
      temp = tempS[i][k];
      tempS[i][k] = temp - s * (tempS[i][l] + tau * temp);
      tempS[i][l] = tempS[i][l] + s * (temp - tau * tempS[i][l]);
    }
    for(int i=k+1; i<l; ++i) {
      temp = tempS[k][i];
      tempS[k][i] = temp - s * (tempS[i][l] + tau * tempS[k][i]);
      tempS[i][l] = tempS[i][l] + s * (temp - tau * tempS[i][l]);
    }
    for(int i=l+1; i<N; ++i) {
      temp = tempS[k][i];
      tempS[k][i] = temp - s * (tempS[l][i] + tau * temp);
      tempS[l][i] = tempS[l][i] + s * (temp - tau * tempS[l][i]);
    }
    for(int i=0; i<N; ++i) {
      temp = P[i][k];
      P[i][k] = temp - s * (P[i][l] + tau * P[i][k]);
      P[i][l] = P[i][l] + s * (temp - tau * P[i][l]);
    }
  }
}

if(small == 0) {
  std::cout << "Metode Jacobi tidak sesuai." << std::endl;
  for(int i=0; i<N; ++i) {
    eigenvalues[i] = tempS[i][i];
  }
}

// urutkan berdasarkan eigenvalues
// first = eigenvalues
// second = P
for(int i=0; i<N-1; ++i) {
  int index = i;
  double value = eigenvalues[i];
  for(int j=i+1; j<N; ++j) {
    if(eigenvalues[j] > value) {
      index = j;
      value = eigenvalues[j];
    }
  }
```

```
    if(index != i) {
      std::swap(eigenvalues[i], eigenvalues[index]);
      for(int r=0; r<N; ++r) {
        std::swap(P[r][i], P[r][index]);
      }
    }
  }
  for(int r=0; r<N; ++r) {
    for(int c=0; c<N; ++c) {
      V[r][c] = P[c][r];
    }
  }
}

void calculate_eigenfaces() {
  double **Urow, **eigenface;
  Urow = (double**) malloc(1*sizeof(double*));
  for(int x=0; x<1; x++) {
    Urow[x] = (double*) malloc(M*sizeof(double));
    memset(Urow[x],0,M*sizeof(double));
  }
  eigenface = (double**) malloc(1*sizeof(double*));
  for(int x=0; x<1; x++) {
    eigenface[x] = (double*) malloc(M*sizeof(double));
    memset(eigenface[x],0,M*sizeof(double));
  }

  for(int r=0; r<Eigenfaces; ++r) {
    for(int c=0; c<M; ++c) {
      eigenface[0][c] = 0;
    }
    for(int re=0; re<1; re++) {
      for(int c=0; c<M; ++c) {
        for(int k=0; k<N; ++k) {
          eigenface[re][c] += V[r][k] * A[k][c];
        }
      }
    }

    for(int c=0; c<M; ++c) {
      U[r][c] = eigenface[0][c];
    }

    double norm = 0;
    for(int i=0; i<M; i++) {
      norm += pow(U[r][i], 2);
    }
    norm = sqrt(norm);
    for(int i=0; i<M; i++) {
      U[r][i] /= norm;
    }

    // output eigenface

    // eigenface <- scale(U[r])
    // temukan minimum dan maksimum saat ini
```

```
    for(int c=0; c<M; ++c) {
      Urow[0][c] = U[r][c];
    }

    double min = 0, max = 255;
    double m_min = Urow[0][0];
    double m_max = Urow[0][0];
    for(int rs=0; rs<1; ++rs) {
        for(int c=0; c<M; ++c) {
            if(Urow[rs][c] < m_min) {
                m_min = Urow[rs][c];
            }
            if(Urow[rs][c] > m_max) {
                m_max = Urow[rs][c];
            }
        }
    }

    double old_range = m_max - m_min;
    double new_range = max - min;

    // buat matriks baru dengan elemen berskala
    for(int re=0; re<1; ++re) {
      for(int c=0; c<M; ++c) {
        eigenface[re][c] = (Urow[re][c] -
 m_min) * new_range / old_range + min;
      }
    }

    std::ostringstream filename;
    filename << "output/eigenfaces/" << r << ".pgm";

    // write pgm
    std::ofstream image_file(filename.str().c_str());
    image_file << "P2" << std::endl << Width << std::endl << Heigh
t << std::endl << MaxValue << std::endl;
    for(int c=0; c<M; ++c) {
      int val = eigenface[0][c];
      if(val < 0) {
        val = 0;
      }
      image_file << val << " ";
    }
    image_file.close();
  }

  free(Urow);
  free(eigenface);
}

void calculate_weight() {
  double **Arow, **ArowTrans;
  Arow = (double**) malloc(1*sizeof(double*));
  for(int x=0; x<1; x++) {
    Arow[x] = (double*) malloc(M*sizeof(double));
    memset(Arow[x],0,M*sizeof(double));
```

```
  }
  ArowTrans = (double**) malloc(M*sizeof(double*));
  for(int x=0; x<M; x++) {
    ArowTrans[x] = (double*) malloc(1*sizeof(double));
    memset(ArowTrans[x],0,1*sizeof(double));
  }

  for(int re=0; re<Eigenfaces; ++re) {
    for(int ce=0; ce<N; ++ce) {
      double befW = 0;
      for(int c=0; c<M; ++c) {
        Arow[0][c] = A[ce][c];
      }

      // A[ce] transpose
      for(int r=0; r<M; ++r) {
        for(int c=0; c<1; ++c) {
          ArowTrans[r][c] = Arow[c][r];
        }
      }

      // befW = U[re] * A[ce] transpose
      for(int r=0; r<1; r++) {
        for(int c=0; c<1; ++c) {
          for(int k=0; k<M; ++k) {
            befW += U[re][k] * ArowTrans[k][c];
          }
        }
      }

      W[re][ce] = befW;
    }
  }

  free(Arow);
  free(ArowTrans);
}

void calculate_accuracy() {
  double **Wx, **Xtrans;
  Wx = (double**) malloc(Eigenfaces*sizeof(double*));
  for(int x=0; x<Eigenfaces; x++) {
    Wx[x] = (double*) malloc(1*sizeof(double));
    memset(Wx[x],0,1*sizeof(double));
  }
  Xtrans = (double**) malloc(M*sizeof(double*));
  for(int x=0; x<M; x++) {
    Xtrans[x] = (double*) malloc(1*sizeof(double));
    memset(Xtrans[x],0,1*sizeof(double));
  }

  double accuracy = 0;
  for(int i=1; i<=N; ++i) {
    // baca sample gambar
    std::stringstream filesample;
```

```
    filesample << DataPath << "s" << i << "/" << SampleName << ".p
gm";
    // filesample << DataPath << i << "_" << SampleName << ".pgm";
    std::ifstream imagesample(filesample.str().c_str());

    if (imagesample.is_open()) {
      // baca foto
      std::string line;
      getline(imagesample, line); // Skip P2 line
      getline(imagesample, line); // Skip width line
      getline(imagesample, line); // Skip height line
      getline(imagesample, line); // Skip max value line

      int val;
      int x = 0;
      while(imagesample >> val) {
      X[0][x] = val;
        x++;
      }

      imagesample.close();
    } else {
      std::cout << "Image was not opened.";
    }

    // pengenalan (X, B, U, W)
    // kurangi gambar rata-rata
    for(int c=0; c<M; ++c) {
      X[0][c] -= B[0][c];
      if(X[0][c] < 0) {
        X[0][c] = 0;
      }
    }
    // temukan bobot
    for(int rw=0; rw<Eigenfaces; ++rw) {
      double befWx = 0;

      // transpose X
      for(int r=0; r<M; ++r) {
        for(int c=0; c<1; ++c) {
          Xtrans[r][c] = X[c][r];
        }
      }

      // befWx = U[rw] * X transpose
      for(int r=0; r<1; r++) {
        for(int c=0; c<1; ++c) {
          for(int k=0; k<M; ++k) {
            befWx += U[rw][k] * Xtrans[k][c];
          }
        }
      }
      Wx[rw][0] = befWx;
    }

    // temukan wajah terdekat dari set pelatihan
```

```
    double min_distance = 0;
    int image_number = 0;
    for(int img=0; img<N; ++img) {
      double distance = 0;
      for(int eface=0; eface<Eigenfaces; ++eface) {
        distance += fabs(W[eface][img] - Wx[eface][0]);
      }
      if(distance < min_distance || img == 0) {
        min_distance = distance;
        image_number = img;
      }
    }

    // std::cout << i << ". " << image_number + 1 << std::endl;
    if(i == image_number + 1) {
      accuracy = accuracy + 1;
    }
  }

  std::cout << "Accuracy : " << std::fixed << std::setprecision(2)
 << accuracy / N << std::endl;

  free(Wx);
  free(Xtrans);
}

int main(int argc, char *argv[]) {
  srand(time(NULL));
  first = omp_get_wtime();

  // A berisi gambar sebagai baris. A adalah NxM, [A] i, j adalah
nilai piksel ke-j gambar ke-i.
  A = (double**) malloc(N*sizeof(double*));
  for(int x=0; x<N; x++) {
    A[x] = (double*) malloc(M*sizeof(double));
    memset(A[x],0,M*sizeof(double));
  }

  // baca data latih
  t = omp_get_wtime();
  read_training_data();
  t = omp_get_wtime() - t;
  printf("Execution time read training data : %.2fms\n", t*1000);

  // B berisi citra rata-
rata. B adalah matriks 1xM, [B] 0, j adalah nilai piksel ke-
j dari citra rata-rata.
  B = (double**) malloc(1*sizeof(double*));
  for(int x=0; x<1; x++) {
    B[x] = (double*) malloc(M*sizeof(double));
    memset(B[x],0,M*sizeof(double));
  }

  create_mean_image();
  normalized();
```

```
// transpose matriks A
Atrans = (double**) malloc(M*sizeof(double*));
for(int x=0; x<M; x++) {
  Atrans[x] = (double*) malloc(N*sizeof(double));
  memset(Atrans[x],0,N*sizeof(double));
}
t = omp_get_wtime();
transpose_matrixA();
t = omp_get_wtime() - t;
printf("Execution time transpose matrix A : %.2fms\n", t*1000);

// matriks covarian [A * Atranspose]
// S -> M*M
// tempS -> M*M
S = (double**) malloc(N*sizeof(double*));
tempS = (double**) malloc(N*sizeof(double*));
P = (double**) malloc(N*sizeof(double*));
V = (double**) malloc(N*sizeof(double*));
for(int x=0; x<N; x++) {
  S[x] = (double*) malloc(N*sizeof(double));
  tempS[x] = (double*) malloc(N*sizeof(double));
  P[x] = (double*) malloc(N*sizeof(double));
  V[x] = (double*) malloc(N*sizeof(double));
  memset(S[x],0,N*sizeof(double));
  memset(tempS[x],0,N*sizeof(double));
  memset(P[x],0,N*sizeof(double));
  memset(V[x],0,N*sizeof(double));
}

t = omp_get_wtime();
// serial
get_covariant_matrix();
t = omp_get_wtime() - t;
printf("Execution time get covariant matrix : %.2fms\n", t*1000);

t = omp_get_wtime();
calculate_eigenvalues();
free(tempS);
t = omp_get_wtime() - t;
printf("Execution time get eigenvalues : %.2fms\n", t*1000);

// temukan eigenface
U = (double**) malloc(Eigenfaces*sizeof(double*));
for(int x=0; x<Eigenfaces; x++) {
  U[x] = (double*) malloc(M*sizeof(double));
  memset(U[x],0,M*sizeof(double));
}

t = omp_get_wtime();
calculate_eigenfaces();
t = omp_get_wtime() - t;
printf("Execution time get eigenfaces : %.2fms\n", t*1000);

// temukan bobot
W = (double**) malloc(Eigenfaces*sizeof(double*));
for(int x=0; x<Eigenfaces; x++) {
```

```
    W[x] = (double*) malloc(N*sizeof(double));
    memset(W[x],0,N*sizeof(double));
  }

  t = omp_get_wtime();
  calculate_weight();
  t = omp_get_wtime() - t;
  printf("Execution time get weight : %.2fms\n", t*1000);

  // menghitung akurasi
  X = (double**) malloc(1*sizeof(double*));
  for(int x=0; x<1; x++) {
    X[x] = (double*) malloc(M*sizeof(double));
    memset(X[x],0,M*sizeof(double));
  }

  t = omp_get_wtime();
  calculate_accuracy();
  t = omp_get_wtime() - t;
  printf("Execution time get accuracy : %.2fms\n", (float)(t)/CLOC
KS_PER_SEC*1000);
  first = omp_get_wtime() - first;
  printf("Total Execution time : %.2fms\n", first*1000);

  free(X);
  free(A);
  free(Atrans);
  free(B);
  free(S);
  free(P);
  free(U);
  free(V);
  free(W);
  return 0;
}
```

Penambahan Baris Kode untuk menentukan eigenface

```
.
.
int main(int argc, char *argv[]) {
 if(argc == 2){
    Eigenfaces = atoi(argv[1]);
  }
.
.
```

Baris kode program paralel

```
#include <iostream>
#include <iomanip>
#include <sstream>
#include <fstream>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <algorithm>
#include <omp.h>
#include <time.h>

#define MIN(a,b) (((a)<(b))?(a):(b))
const int Faces = 40;
const int Samples = 9;
const int Width = 493;    //  92 | 296 | 394 | 493
const int Height = 600;   // 112 | 360 | 480 | 600
int Eigenfaces = 28;
const std::string DataPath = "face600/";  // faces | face360 | fac
e480 | face600
const int N = Faces;
const int M = Width * Height;
const std::string SampleName = "10";
const int MaxValue = 255;
double **facearray, **A, **B, **S, **V, **U, **W, **X, **P, **Atra
ns, **tempS;
double first, t, tsc;

void read_training_data() {
  facearray = (double**) malloc(Samples*sizeof(double*));
  for(int x=0; x<Samples; x++) {
    facearray[x] = (double*) malloc(M*sizeof(double));
    memset(facearray[x],0,M*sizeof(double));
  }
  for(int face=0; face<Faces; ++face) {
    // perulangan setiap foto
    for(int sample=0; sample<Samples; ++sample) {
      std::stringstream filename;
      // filename << DataPath << "s" << face + 1 << "/" << sample
 + 1 << ".pgm";
      filename << DataPath << face + 1 << "_" << sample  + 1 << ".
pgm";
      std::ifstream image(filename.str().c_str());

      if (image.is_open()) {
        // baca foto

        std::string line;
        getline(image, line); // Skip P2 line
        getline(image, line); // Skip width line
        getline(image, line); // Skip height line
        getline(image, line); // Skip max value line

        int val;
        int x = 0;
```

```
        while(image >> val) {
        facearray[sample][x] = val;
          x++;
        }

        image.close();
      } else {
        std::cout << "Image was not opened.";
      }
    }

    // mencari citra rata-rata
    for(int x=0; x<M; ++x) {
      double sum = 0;
      for(int y=0; y<Samples; ++y) {
        sum += facearray[y][x];
      }
      A[face][x] = sum/Samples;
    }
  }
  free(facearray);
}

void create_mean_image() {
  // temukan mean image
  tsc = omp_get_wtime();
  for(int c=0; c<M; ++c) {
    double sum = 0;
    for(int r=0; r<N; ++r) {
      sum += A[r][c];
    }
    B[0][c] = sum / N;
  }
  tsc = omp_get_wtime() - tsc;
  printf("Execution time create mean image : %.2fms\n", tsc*1000);

  // output citra rata-rata
  std::stringstream filename;
  filename << "output/meanimage.pgm";
  std::ofstream image_file(filename.str().c_str());
  image_file << "P2" << std::endl << Width << std::endl << Height
<< std::endl << MaxValue << std::endl;
  for(int x=0; x<M; ++x) {
    int val = B[0][x];
    if(val < 0) {
      val = 0;
    }
    image_file << val << " ";
  }
  image_file.close();
}

void normalized() {
  // kurangkan mean dari setiap gambar
  tsc = omp_get_wtime();
  for(int r=0; r<N; ++r) {
```

```
    for(int c=0; c<M; ++c) {
      A[r][c] -= B[0][c];
      if(A[r][c] < 0) {
        A[r][c] = 0;
      }
    }
  }
  tsc = omp_get_wtime() - tsc;
  printf("Execution time mean subtraction : %.2fms\n", tsc*1000);

  // output citra normalisasi
  for(int x=0; x<N; ++x) {
    std::ostringstream filename;
    filename << "output/normalized/" << x << ".pgm";
    std::ofstream image_file(filename.str().c_str());
    image_file << "P2" << std::endl << Width << std::endl << Heigh
t << std::endl << MaxValue << std::endl;
    for(int y=0; y<M; ++y) {
      int val = A[x][y];
      if(val < 0) {
        val = 0;
      }
      image_file << val << " ";
    }
    image_file.close();
  }
}

void transpose_matrixA() {
  for(int r=0; r<M; ++r) {
    for(int c=0; c<N; ++c) {
      Atrans[r][c] = A[c][r];
    }
  }
}

void get_covariant_matrix() {
  int r;
  #pragma omp parallel
  {
    #pragma omp single nowait
    for(r=0; r<N; ++r) {
      #pragma omp task firstprivate(r)
      for(int c=0; c<N; ++c) {
        S[r][c] = 0;
        for(int k=0; k<M; ++k) {
          S[r][c] += A[r][k] * Atrans[k][c];
        }
      }
    }
  }
}

void calculate_eigenvalues() {
  // eigenvector pada matriks covarian
  // eigensystem(S).second.transpose()
```

```
for(int r=0; r<N; ++r) {
  for(int c=0; c<N; ++c) {
    if (c == r) {
      P[c][r] = 1;
    }
  }
}
// std::cout<<"P[0][2] : "<<P[0][2]<<std::endl;
int max_rotation = 5 * N * N;
for(int i=0; i<N; ++i) {
  for(int j=0; j<N; ++j) {
    tempS[i][j] = S[i][j];
  }
}

double *eigenvalues;
eigenvalues = (double*) malloc(N*sizeof(double));
int small = 0;

for(int it=0; it<max_rotation; ++it) {
  double max = 0;
  int k, l;
  // temukan elemen off-diagonal terbesar
  for(int r=0; r<N-1; ++r) {
    for(int c=r+1; c<N; ++c) {
      if (fabs(tempS[r][c]) >= max) {
        max = fabs(tempS[r][c]);
        k = r;
        l = c;
      }
    }
  }

  if(max < 1.0e-12) {
    for(int i=0; i<N; ++i) {
      eigenvalues[i] = tempS[i][i];
    }
    // normalisasi P
    for(int c=0; c<N; ++c) {
      double length = 0;
      for(int r=0; r<N; ++r) {
        length += P[r][c] * P[r][c];
      }
      for(int r=0; r<N; ++r) {
        P[r][c] = P[r][c] / length;
      }
    }
    small = 1;
    break;
  }
  else {
    // melakukan rotasi
    double diff = tempS[l][l] - tempS[k][k];

    double t;
    if(fabs(tempS[k][l]) < fabs(diff)*1.0e-36) {
```

```
      t = tempS[k][l] / diff;
    }
    else {
      double phi = diff / (2.0 * tempS[k][l]);
      t = 1.0 / (fabs(phi) + sqrt(phi * phi + 1.0));
      if(phi < 0) {
        t = -t;
      }
    }
    double c = 1.0 / sqrt(t * t + 1.0);
    double s = t * c;
    double tau = s / (1.0 + c);
    double temp = tempS[k][l];
    tempS[k][l] = 0;
    tempS[k][k] = tempS[k][k] - t * temp;
    tempS[l][l] = tempS[l][l] + t * temp;

    for(int i=0; i<k; ++i) {
      temp = tempS[i][k];
      tempS[i][k] = temp - s * (tempS[i][l] + tau * temp);
      tempS[i][l] = tempS[i][l] + s * (temp - tau * tempS[i][l]);
    }
    for(int i=k+1; i<l; ++i) {
      temp = tempS[k][i];
      tempS[k][i] = temp - s * (tempS[i][l] + tau * tempS[k][i]);
      tempS[i][l] = tempS[i][l] + s * (temp - tau * tempS[i][l]);
    }
    for(int i=l+1; i<N; ++i) {
      temp = tempS[k][i];
      tempS[k][i] = temp - s * (tempS[l][i] + tau * temp);
      tempS[l][i] = tempS[l][i] + s * (temp - tau * tempS[l][i]);
    }
    for(int i=0; i<N; ++i) {
      temp = P[i][k];
      P[i][k] = temp - s * (P[i][l] + tau * P[i][k]);
      P[i][l] = P[i][l] + s * (temp - tau * P[i][l]);
    }
  }
}

if(small == 0) {
  std::cout << "Metode Jacobi tidak sesuai." << std::endl;
  for(int i=0; i<N; ++i) {
    eigenvalues[i] = tempS[i][i];
  }
}

// urutkan berdasarkan eigenvalues
// first = eigenvalues
// second = P
for(int i=0; i<N-1; ++i) {
  int index = i;
  double value = eigenvalues[i];
  for(int j=i+1; j<N; ++j) {
    if(eigenvalues[j] > value) {
      index = j;
```

```
        value = eigenvalues[j];
      }
    }
    if(index != i) {
      std::swap(eigenvalues[i], eigenvalues[index]);
      for(int r=0; r<N; ++r) {
        std::swap(P[r][i], P[r][index]);
      }
    }
  }
  for(int r=0; r<N; ++r) {
    for(int c=0; c<N; ++c) {
      V[r][c] = P[c][r];
    }
  }
}

void calculate_eigenfaces() {
  double **Urow, **eigenface;
  Urow = (double**) malloc(1*sizeof(double*));
  for(int x=0; x<1; x++) {
    Urow[x] = (double*) malloc(M*sizeof(double));
    memset(Urow[x],0,M*sizeof(double));
  }
  eigenface = (double**) malloc(1*sizeof(double*));
  for(int x=0; x<1; x++) {
    eigenface[x] = (double*) malloc(M*sizeof(double));
    memset(eigenface[x],0,M*sizeof(double));
  }

  int r;
  #pragma omp parallel
  {
    #pragma omp single nowait
    for(r=0; r<Eigenfaces; ++r) {
      #pragma omp task firstprivate(r)
      for(int c=0; c<M; ++c) {
        U[r][c] = 0;
        for(int k=0; k<N; ++k) {
          U[r][c] += V[r][k] * A[k][c];
        }
      }
    }
  }

  for(int r=0; r<Eigenfaces; ++r) {
    double norm = 0;
    for(int i=0; i<M; i++) {
      norm += pow(U[r][i], 2);
    }
    norm = sqrt(norm);
    for(int i=0; i<M; i++) {
      U[r][i] /= norm;
    }

    // output eigenface
```

```
    // eigenface <- scale(U[r])
    // temukan minimum dan maksimum saat ini
    for(int c=0; c<M; ++c) {
      Urow[0][c] = U[r][c];
    }

    double min = 0, max = 255;
    double m_min = Urow[0][0];
    double m_max = Urow[0][0];
    for(int rs=0; rs<1; ++rs) {
        for(int c=0; c<M; ++c) {
            if(Urow[rs][c] < m_min) {
                m_min = Urow[rs][c];
            }
            if(Urow[rs][c] > m_max) {
                m_max = Urow[rs][c];
            }
        }
    }

    double old_range = m_max - m_min;
    double new_range = max - min;

    // buat matriks baru dengan elemen berskala
    for(int re=0; re<1; ++re) {
      for(int c=0; c<M; ++c) {
        eigenface[re][c] = (Urow[re][c] -
 m_min) * new_range / old_range + min;
      }
    }

    std::ostringstream filename;
    filename << "output/eigenfaces/" << r << ".pgm";

    // write pgm
    std::ofstream image_file(filename.str().c_str());
    image_file << "P2" << std::endl << Width << std::endl << Heigh
t << std::endl << MaxValue << std::endl;
    for(int c=0; c<M; ++c) {
      int val = eigenface[0][c];
      if(val < 0) {
        val = 0;
      }
      image_file << val << " ";
    }
    image_file.close();
  }

  free(Urow);
  free(eigenface);
}

void calculate_weight() {
  int r;
  #pragma omp parallel
```

```
  {
    #pragma omp single nowait
    for(r=0; r<Eigenfaces; ++r) {
      #pragma omp task firstprivate(r)
      for(int c=0; c<N; ++c) {
        W[r][c] = 0;
        for(int k=0; k<M; ++k) {
          W[r][c] += U[r][k] * Atrans[k][c];
        }
      }
    }
  }
}

void calculate_accuracy() {
  double **Wx, **Xtrans;
  Wx = (double**) malloc(Eigenfaces*sizeof(double*));
  for(int x=0; x<Eigenfaces; x++) {
    Wx[x] = (double*) malloc(1*sizeof(double));
    memset(Wx[x],0,1*sizeof(double));
  }
  Xtrans = (double**) malloc(M*sizeof(double*));
  for(int x=0; x<M; x++) {
    Xtrans[x] = (double*) malloc(1*sizeof(double));
    memset(Xtrans[x],0,1*sizeof(double));
  }

  double accuracy = 0;
  for(int i=1; i<=N; ++i) {
    // baca sample gambar
    std::stringstream filesample;
    // filesample << DataPath << "s" << i << "/" << SampleName <<
".pgm";
    filesample << DataPath << i << "_" << SampleName << ".pgm";
    std::ifstream imagesample(filesample.str().c_str());

    if (imagesample.is_open()) {
      // baca foto
      std::string line;
      getline(imagesample, line); // Skip P2 line
      getline(imagesample, line); // Skip width line
      getline(imagesample, line); // Skip height line
      getline(imagesample, line); // Skip max value line

      int val;
      int x = 0;
      while(imagesample >> val) {
      X[0][x] = val;
        x++;
      }

      imagesample.close();
    } else {
      std::cout << "Image was not opened.";
    }
```

```
    // pengenalan (X, B, U, W)
    // kurangi gambar rata-rata
    for(int c=0; c<M; ++c) {
      X[0][c] -= B[0][c];
      if(X[0][c] < 0) {
        X[0][c] = 0;
      }
    }
    // temukan bobot
    for(int rw=0; rw<Eigenfaces; ++rw) {
      double befWx = 0;

      // transpose X
      for(int r=0; r<M; ++r) {
        for(int c=0; c<1; ++c) {
          Xtrans[r][c] = X[c][r];
        }
      }

      // befWx = U[rw] * X transpose
      for(int r=0; r<1; r++) {
        for(int c=0; c<1; ++c) {
          for(int k=0; k<M; ++k) {
            befWx += U[rw][k] * Xtrans[k][c];
          }
        }
      }
      Wx[rw][0] = befWx;
    }

    // temukan wajah terdekat dari set pelatihan
    double min_distance = 0;
    int image_number = 0;
    for(int img=0; img<N; ++img) {
      double distance = 0;
      for(int eface=0; eface<Eigenfaces; ++eface) {
        distance += fabs(W[eface][img] - Wx[eface][0]);
      }
      if(distance < min_distance || img == 0) {
        min_distance = distance;
        image_number = img;
      }
    }

    // std::cout << i << ". " << image_number + 1 << std::endl;
    if(i == image_number + 1) {
      accuracy = accuracy + 1;
    }
  }

  std::cout << "Accuracy : " << std::fixed << std::setprecision(2)
 << accuracy / N << std::endl;

  free(Wx);
  free(Xtrans);
}
```

```
int main(int argc, char *argv[]) {
  int thread = 2;
  if(argc == 2){
    thread = atoi(argv[1]);
  }

  srand(time(NULL));
  first = omp_get_wtime();

  omp_set_num_threads(thread);

  // A berisi gambar sebagai baris. A adalah NxM, [A] i, j adalah
nilai piksel ke-j gambar ke-i.
  A = (double**) malloc(N*sizeof(double*));
  for(int x=0; x<N; x++) {
    A[x] = (double*) malloc(M*sizeof(double));
    memset(A[x],0,M*sizeof(double));
  }

  // baca data latih
  t = omp_get_wtime();
  read_training_data();
  t = omp_get_wtime() - t;
  printf("Execution time read training data : %.2fms\n", t*1000);

  // B berisi citra rata-
rata. B adalah matriks 1xM, [B] 0, j adalah nilai piksel ke-
j dari citra rata-rata.
  B = (double**) malloc(1*sizeof(double*));
  for(int x=0; x<1; x++) {
    B[x] = (double*) malloc(M*sizeof(double));
    memset(B[x],0,M*sizeof(double));
  }

  create_mean_image();
  normalized();

  // transpose matriks A
  Atrans = (double**) malloc(M*sizeof(double*));
  for(int x=0; x<M; x++) {
    Atrans[x] = (double*) malloc(N*sizeof(double));
    memset(Atrans[x],0,N*sizeof(double));
  }
  t = omp_get_wtime();
  transpose_matrixA();
  t = omp_get_wtime() - t;
  printf("Execution time transpose matrix A : %.2fms\n", t*1000);

  // matriks covarian [A * Atranspose]
  // S -> M*M
  // tempS -> M*M
  S = (double**) malloc(N*sizeof(double*));
  tempS = (double**) malloc(N*sizeof(double*));
  P = (double**) malloc(N*sizeof(double*));
  V = (double**) malloc(N*sizeof(double*));
```

```
for(int x=0; x<N; x++) {
  S[x] = (double*) malloc(N*sizeof(double));
  tempS[x] = (double*) malloc(N*sizeof(double));
  P[x] = (double*) malloc(N*sizeof(double));
  V[x] = (double*) malloc(N*sizeof(double));
  memset(S[x],0,N*sizeof(double));
  memset(tempS[x],0,N*sizeof(double));
  memset(P[x],0,N*sizeof(double));
  memset(V[x],0,N*sizeof(double));
}

t = omp_get_wtime();
// serial
get_covariant_matrix();
t = omp_get_wtime() - t;
printf("Execution time get covariant matrix : %.2fms\n", t*1000);

t = omp_get_wtime();
calculate_eigenvalues();
free(tempS);
t = omp_get_wtime() - t;
printf("Execution time get eigenvalues : %.2fms\n", t*1000);

// temukan eigenface
U = (double**) malloc(Eigenfaces*sizeof(double*));
for(int x=0; x<Eigenfaces; x++) {
  U[x] = (double*) malloc(M*sizeof(double));
  memset(U[x],0,M*sizeof(double));
}

t = omp_get_wtime();
calculate_eigenfaces();
t = omp_get_wtime() - t;
printf("Execution time get eigenfaces : %.2fms\n", t*1000);

// temukan bobot
W = (double**) malloc(Eigenfaces*sizeof(double*));
for(int x=0; x<Eigenfaces; x++) {
  W[x] = (double*) malloc(N*sizeof(double));
  memset(W[x],0,N*sizeof(double));
}

t = omp_get_wtime();
calculate_weight();
t = omp_get_wtime() - t;
printf("Execution time get weight : %.2fms\n", t*1000);

// menghitung akurasi
X = (double**) malloc(1*sizeof(double*));
for(int x=0; x<1; x++) {
  X[x] = (double*) malloc(M*sizeof(double));
  memset(X[x],0,M*sizeof(double));
}

t = omp_get_wtime();
calculate_accuracy();
```

```
  t = omp_get_wtime() - t;
  printf("Execution time get accuracy : %.2fms\n", (float)(t)/CLOC
KS_PER_SEC*1000);
  first = omp_get_wtime() - first;
  printf("Total Execution time : %.2fms\n", first*1000);

  free(X);
  free(A);
  free(Atrans);
  free(B);
  free(S);
  free(P);
  free(U);
  free(V);
  free(W);
  return 0;
}
```

LAMPIRAN II DATA HASIL PROGRAM

Nilai Akurasi setiap eigenface

Number of Eigenface	Recognition Rate
1	3%
2	17%
3	38%
4	50%
5	55%
6	47%
7	62%
8	60%
9	65%
10	75%
11	70%
12	72%
13	70%
14	80%
15	80%
16	82%
17	88%
18	85%
19	88%
20	90%

Number of Eigenface	Recognition Rate
21	93%
22	93%
23	93%
24	93%
25	95%
26	95%
27	95%
28	100%
29	97%
30	97%
31	97%
32	93%
33	93%
34	95%
35	93%
36	95%
37	95%
38	95%
39	95%
40	95%

Waktu Eksekusi pada gambar berukuran 92x112

Uji	Fungsi	Waktu eksekusi (detik)				
		Serial	Paralel (jumlah thread)			
			2	4	6	8
1	read data	2,28	4,72	3,73	4,04	3,62
	mean image	0,02	0,02	0,02	0,02	0,02
	subtracted	0,04	0,04	0,04	0,04	0,04
	transpose	0,03	0,02	0,02	0,02	0,02
	covariance	3,61	1,55	0,81	0,67	0,55
	eigenvalue	0,10	0,09	0,09	0,09	0,09
	eigenfaces	1,35	0,61	0,37	0,31	0,28
	weight	1,47	1,13	0,57	0,44	0,48
	total	9,06	8,35	5,82	5,79	5,25
2	read data	2,26	3,60	3,32	4,93	4,46
	mean image	0,02	0,02	0,02	0,02	0,02
	subtracted	0,05	0,04	0,04	0,04	0,04
	transpose	0,03	0,02	0,02	0,02	0,02
	covariance	3,66	1,57	0,81	0,59	0,55
	eigenvalue	0,10	0,09	0,09	0,09	0,09
	eigenfaces	1,35	0,61	0,37	0,31	0,27
	weight	1,58	1,14	0,59	0,48	0,37
	total	9,20	7,24	5,42	6,64	5,99
3	read data	2,28	3,55	4,57	4,75	3,85
	mean image	0,02	0,02	0,02	0,02	0,02
	subtracted	0,05	0,04	0,04	0,04	0,04
	transpose	0,03	0,02	0,02	0,02	0,02
	covariance	3,62	1,55	0,81	0,59	0,56
	eigenvalue	0,10	0,09	0,09	0,09	0,09
	eigenfaces	1,35	0,61	0,38	0,31	0,28
	weight	1,50	1,13	0,59	0,44	0,47
	total	9,10	7,17	6,69	6,41	5,49
4	read data	2,28	4,55	4,42	4,57	4,17
	mean image	0,02	0,02	0,02	0,02	0,02
	subtracted	0,05	0,04	0,04	0,04	0,04
	transpose	0,03	0,02	0,02	0,02	0,02
	covariance	3,66	1,56	0,81	0,61	0,56
	eigenvalue	0,10	0,09	0,09	0,09	0,09
	eigenfaces	1,34	0,65	0,37	0,32	0,29
	weight	1,54	1,17	0,59	0,45	0,40
	total	9,16	8,26	6,53	6,28	5,75

Uji	Fungsi	Waktu eksekusi (detik)				
		Serial	Paralel (jumlah thread)			
			2	4	6	8
5	read data	2,28	3,79	4,88	3,89	3,69
	mean image	0,02	0,02	0,02	0,02	0,02
	subtracted	0,05	0,04	0,04	0,04	0,04
	transpose	0,03	0,02	0,02	0,02	0,02
	covariance	3,65	1,59	0,80	0,59	0,56
	eigenvalue	0,10	0,10	0,09	0,09	0,09
	eigenfaces	1,34	0,64	0,38	0,31	0,29
	weight	1,47	1,11	0,58	0,44	0,46
	total	9,08	7,49	6,98	5,56	5,33

Waktu Eksekusi pada gambar berukuran 296x360

Uji	Fungsi	Waktu eksekusi (detik)				
		Serial	Paralel (jumlah thread)			
			2	4	6	8
1	read data	24,03	23,90	23,36	23,36	23,36
	mean image	0,28	0,29	0,30	0,30	0,29
	subtracted	0,50	0,48	0,53	0,53	0,53
	transpose	0,29	0,29	0,29	0,29	0,29
	covariance	47,32	23,74	12,16	8,52	7,29
	eigenvalue	0,09	0,10	0,10	0,10	0,14
	eigenfaces	15,51	7,40	4,50	4,02	3,81
	weight	25,68	16,71	8,48	6,24	5,41
	total	115,31	74,54	51,19	44,83	42,62
2	read data	23,41	23,39	23,37	23,36	23,60
	mean image	0,27	0,30	0,29	0,29	0,30
	subtracted	0,52	0,53	0,53	0,53	0,53
	transpose	0,27	0,29	0,29	0,29	0,29
	covariance	47,37	23,91	12,15	8,77	7,25
	eigenvalue	0,10	0,10	0,10	0,10	0,14
	eigenfaces	15,55	7,49	4,50	4,02	4,01
	weight	25,69	16,73	8,49	6,31	5,54
	total	114,87	74,20	51,20	45,14	43,13
3	read data	23,00	23,35	23,36	24,08	23,39
	mean image	0,27	0,30	0,30	0,30	0,29
	subtracted	0,52	0,53	0,53	0,53	0,53
	transpose	0,29	0,29	0,29	0,29	0,29
	covariance	47,33	23,87	12,29	8,87	7,07
	eigenvalue	0,09	0,10	0,10	0,10	0,14
	eigenfaces	15,59	7,40	4,50	4,19	3,89
	weight	25,63	16,71	8,49	6,42	5,48
	total	114,34	74,02	51,32	46,23	42,57

Uji	Fungsi	Waktu eksekusi (detik)				
		Serial	Paralel (jumlah thread)			
			2	4	6	8
4	read data	23,07	23,34	23,35	23,34	23,36
	mean image	0,27	0,30	0,30	0,30	0,29
	subtracted	0,49	0,53	0,53	0,53	0,53
	transpose	0,29	0,29	0,29	0,29	0,29
	covariance	47,19	23,87	12,15	8,94	7,15
	eigenvalue	0,09	0,10	0,10	0,10	0,14
	eigenfaces	15,45	7,46	4,51	4,31	3,84
	weight	25,54	16,71	8,49	6,57	5,41
	total	114,07	74,08	51,19	45,87	42,50
5	read data	23,77	23,36	23,34	23,35	23,37
	mean image	0,27	0,30	0,30	0,29	0,30
	subtracted	0,48	0,53	0,53	0,53	0,53
	transpose	0,27	0,29	0,29	0,29	0,29
	covariance	47,12	23,90	12,12	9,05	7,31
	eigenvalue	0,09	0,10	0,10	0,10	0,14
	eigenfaces	15,45	7,40	4,51	4,23	4,03
	weight	25,34	16,73	8,49	6,47	5,55
	total	114,44	74,08	51,14	45,81	43,01

Waktu Eksekusi pada gambar berukuran 394x480

Uji	Fungsi	Waktu eksekusi (detik)				
		Serial	Paralel (jumlah thread)			
			2	4	6	8
1	read data	24,03	23,90	23,36	23,36	23,36
	mean image	0,28	0,29	0,30	0,30	0,29
	subtracted	0,50	0,48	0,53	0,53	0,53
	transpose	0,29	0,29	0,29	0,29	0,29
	covariance	47,32	23,74	12,16	8,52	7,29
	eigenvalue	0,09	0,10	0,10	0,10	0,14
	eigenfaces	15,51	7,40	4,50	4,02	3,81
	weight	25,68	16,71	8,48	6,24	5,41
	total	115,31	74,54	51,19	44,83	42,62
2	read data	23,41	23,39	23,37	23,36	23,60
	mean image	0,27	0,30	0,29	0,29	0,30
	subtracted	0,52	0,53	0,53	0,53	0,53
	transpose	0,27	0,29	0,29	0,29	0,29
	covariance	47,37	23,91	12,15	8,77	7,25
	eigenvalue	0,10	0,10	0,10	0,10	0,14
	eigenfaces	15,55	7,49	4,50	4,02	4,01
	weight	25,69	16,73	8,49	6,31	5,54
	total	114,87	74,20	51,20	45,14	43,13

Uji	Fungsi	Waktu eksekusi (detik)				
		Serial	Paralel (jumlah thread)			
			2	4	6	8
3	read data	23,00	23,35	23,36	24,08	23,39
	mean image	0,27	0,30	0,30	0,30	0,29
	subtracted	0,52	0,53	0,53	0,53	0,53
	transpose	0,29	0,29	0,29	0,29	0,29
	covariance	47,33	23,87	12,29	8,87	7,07
	eigenvalue	0,09	0,10	0,10	0,10	0,14
	eigenfaces	15,59	7,40	4,50	4,19	3,89
	weight	25,63	16,71	8,49	6,42	5,48
	total	114,34	74,02	51,32	46,23	42,57
4	read data	23,07	23,34	23,35	23,34	23,36
	mean image	0,27	0,30	0,30	0,30	0,29
	subtracted	0,49	0,53	0,53	0,53	0,53
	transpose	0,29	0,29	0,29	0,29	0,29
	covariance	47,19	23,87	12,15	8,94	7,15
	eigenvalue	0,09	0,10	0,10	0,10	0,14
	eigenfaces	15,45	7,46	4,51	4,31	3,84
	weight	25,54	16,71	8,49	6,57	5,41
	total	114,07	74,08	51,19	45,87	42,50
5	read data	23,77	23,36	23,34	23,35	23,37
	mean image	0,27	0,30	0,30	0,29	0,30
	subtracted	0,48	0,53	0,53	0,53	0,53
	transpose	0,27	0,29	0,29	0,29	0,29
	covariance	47,12	23,90	12,12	9,05	7,31
	eigenvalue	0,09	0,10	0,10	0,10	0,14
	eigenfaces	15,45	7,40	4,51	4,23	4,03
	weight	25,34	16,73	8,49	6,47	5,55
	total	114,44	74,08	51,14	45,81	43,01

Waktu Eksekusi pada gambar berukuran 493x480

Uji	Fungsi	Waktu eksekusi (detik)				
		Serial	Paralel (jumlah thread)			
			2	4	6	8
1	read data	41,40	41,58	41,59	41,63	40,64
	mean image	0,86	0,78	0,95	0,90	0,92
	subtracted	0,86	0,94	0,94	0,92	0,90
	transpose	0,55	0,55	0,55	0,52	0,53
	covariance	85,41	42,53	21,88	15,35	11,69
	eigenvalue	0,09	0,10	0,10	0,10	0,10
	eigenfaces	33,81	17,15	8,85	6,97	6,53
	weight	46,23	29,68	15,06	11,04	9,21
	total	212,22	135,92	92,53	79,95	73,05

Uji	Fungsi	Waktu eksekusi (detik)				
		Serial	Paralel (jumlah thread)			
			2	4	6	8
2	read data	41,78	41,58	41,60	41,18	41,42
	mean image	1,06	0,80	0,89	0,93	0,77
	subtracted	0,89	0,94	0,94	0,90	0,92
	transpose	0,55	0,55	0,55	0,53	0,55
	covariance	85,29	42,39	21,62	15,26	11,99
	eigenvalue	0,09	0,10	0,10	0,10	0,10
	eigenfaces	31,00	15,28	9,28	7,10	6,81
	weight	46,55	29,88	15,06	10,89	9,38
	total	209,80	134,12	92,64	79,48	74,51
3	read data	41,79	41,57	41,61	41,19	41,97
	mean image	0,88	0,79	0,94	0,94	0,79
	subtracted	0,90	0,94	0,94	0,89	0,93
	transpose	0,55	0,55	0,55	0,55	0,53
	covariance	85,18	42,38	21,65	15,25	12,12
	eigenvalue	0,10	0,10	0,10	0,10	0,10
	eigenfaces	44,15	15,62	9,23	7,88	6,40
	weight	46,26	29,68	15,06	11,28	9,22
	total	222,40	134,24	92,67	80,69	74,64
4	read data	41,71	41,60	41,59	41,57	41,60
	mean image	1,01	0,79	0,96	0,67	0,91
	subtracted	0,94	0,94	0,94	0,92	0,93
	transpose	0,55	0,55	0,55	0,52	0,55
	covariance	85,20	42,38	21,53	15,33	11,90
	eigenvalue	0,10	0,10	0,10	0,10	0,10
	eigenfaces	30,66	15,37	8,65	6,81	7,10
	weight	46,50	29,68	15,06	10,81	9,24
	total	209,26	134,00	92,00	79,31	74,90
5	read data	41,69	41,59	41,59	41,60	42,15
	mean image	1,07	0,82	0,79	0,88	0,94
	subtracted	0,93	0,94	0,92	0,90	0,94
	transpose	0,66	0,55	0,56	0,53	0,55
	covariance	85,32	42,39	21,51	15,54	12,08
	eigenvalue	0,10	0,10	0,10	0,10	0,10
	eigenfaces	30,92	15,48	8,76	7,27	6,69
	weight	46,48	29,68	15,06	10,89	9,31
	total	209,78	134,14	91,86	80,31	75,36

Waktu Eksekusi pada gambar berukuran 493x480

Uji	Fungsi	Waktu eksekusi (detik)				
		Serial	Paralel (jumlah thread)			
			2	4	6	8
1	read data	58,65	57,89	59,51	59,93	60,25
	mean image	1,17	1,51	1,15	1,44	1,49
	subtracted	1,39	1,39	1,42	1,40	1,50
	transpose	0,81	0,83	0,87	0,95	0,88
	covariance	13,59	65,62	33,77	24,18	22,53
	eigenvalue	0,09	0,10	0,10	0,10	0,10
	eigenfaces	42,76	34,08	14,23	12,91	10,83
	weight	71,94	46,17	24,28	17,23	15,91
	total	314,91	212,07	139,37	122,09	117,58
2	read data	59,52	59,16	58,89	60,00	60,11
	mean image	1,69	1,15	1,50	1,50	0,77
	subtracted	1,43	1,42	1,46	1,43	1,51
	transpose	0,83	0,82	0,82	0,85	0,83
	covariance	132,12	65,41	33,63	23,83	23,69
	eigenvalue	0,10	0,10	0,10	0,10	0,10
	eigenfaces	69,10	21,34	15,59	12,06	10,84
	weight	71,32	45,91	23,55	18,63	15,67
	total	340,08	199,30	139,63	122,48	117,68
3	read data	60,09	57,99	59,90	60,12	59,94
	mean image	1,67	1,18	1,22	1,18	0,76
	subtracted	1,43	1,42	1,44	1,42	1,48
	transpose	0,85	0,83	0,82	0,86	0,84
	covariance	133,20	66,09	33,77	24,47	21,51
	eigenvalue	0,10	0,10	0,10	0,10	0,10
	eigenfaces	69,03	22,48	17,13	12,98	10,50
	weight	72,59	46,57	23,55	17,41	16,94
	total	342,91	200,61	141,92	122,51	116,21
4	read data	59,75	59,64	60,44	59,93	60,24
	mean image	1,74	1,21	1,45	1,45	0,80
	subtracted	1,44	1,40	1,43	1,43	1,52
	transpose	0,85	0,84	0,84	0,84	0,85
	covariance	134,13	65,96	34,48	23,65	22,85
	eigenvalue	0,10	0,10	0,10	0,10	0,10
	eigenfaces	71,65	22,46	16,29	12,24	10,50
	weight	72,23	46,34	23,83	18,89	15,81
	total	345,91	201,92	142,87	122,52	116,83

Uji	Fungsi	Waktu eksekusi (detik)				
		Serial	Paralel (jumlah thread)			
			2	4	6	8
5	read data	60,30	59,12	60,02	60,13	60,87
	mean image	1,71	1,17	1,52	1,20	0,79
	subtracted	1,43	1,43	1,41	1,43	1,47
	transpose	0,84	0,84	0,82	0,86	0,84
	covariance	133,43	65,89	33,65	24,23	21,60
	eigenvalue	0,09	0,10	0,10	0,10	0,10
	eigenfaces	69,41	32,29	13,04	11,92	10,91
	weight	72,81	46,60	23,56	18,76	15,35
	total	343,96	211,36	138,17	122,70	116,11