Final Questions
==========

Eigenfaces
----

* Given that forming the covariance matrix with 500 images takes approximately 10 seconds, how long (approximately) would forming the covariance matrix take if we used all 262,781 images in the data file?

Since time to compute depends linearly on the number of images, it would take 262781*10/500 seconds to form the covariance matrix if all images were used.

Sequential Optimization
----

* What optimizations did you apply to improve eigenfaces_opt? Explain why these optimizations would result in better performance.

Changed the ordering of loops, with inner loop iterating over columns of the desired matrix. This allows writing along the rows of the row-major matrix A. These elements are contiguous.

Also, hoisted the reused variable z[k](i) out of the inner (j, i.e. column index of the matrix C) loop, where the index i corresponds to the rows of C. This allows x(i) to be stored in the register and doesn't need load from the memory operation.

```
// Optimize me (sequential)
void outer(Matrix& A, const Vector& x) {
  for (size_t i = 0; i < A.num_rows(); ++i) {
    double t = x(i);
    for (size_t j = 0; j < A.num_cols(); ++j) {
      A(i, j) += t*x(j);
    }
  }
}
```

Parallelization
---

* How did you parallelize your code? On 5000 faces, how much speedup were you able to get on 2, 4, 10, and 20 cores (cpus-per-task) comparing with 1 core case? Does the speedup improve if you load 15000 faces rather than 5000 faces?

The outer loop (index i) is parallelized using # pragma omp parallel for

```
// Basic (original) outer -- Optimize and parallelize me
void outer(Matrix& A, const Vector& x) {
  #pragma omp parallel for
  for (size_t i = 0; i < A.num_rows(); ++i) {
    double t = x(i);
    for (size_t j = 0; j < A.num_cols(); ++j) {
      A(i, j) += t * x(j);
    }
  }
}
```

Considerable improvement on adding more cores. Rate of improvement decreases on adding more cores.
5000 faces: 1 core- 3.17868 GFlops, 2 cores- 7.29067 GFlops, 4 cores- 15.1549 GFlops, 10 cores- 22.2184 GFlops, 20 cores- 38.2374 GFlops

No improvement seen over 5000 images with 15000 images-

15000 faces: 10 cores- 22.4218 GFlops, 20 Cores- 39.0499 GFlops

Blocking (AMATH 583)
---

* Explain your blocking approach. On 5000 faces and 10 (or 20) cores (say), how much speedup were you able to get over the non-blocked case? How much speedup were you able to get over the non-blocked case for 50000 face?

The C matrix is split into blocks of rows; each block is updated by a separate thread.

```
#pragma omp parallel
  {
  size_t tid = omp_get_thread_num();
  size_t parts = omp_get_num_threads();
  size_t blocksize = C.num_rows()/parts;
  size_t begin = tid*blocksize;
  size_t end = (tid+1)*blocksize;

  if (tid == parts - 1) {
     end = C.num_rows();
  }


  for (size_t k = 0; k < z.size(); ++k) {
      #pragma omp parallel for
      for (size_t i = begin; i < end; ++i) {
        double t = z[k](i);
        for (size_t j = 0; j < C.num_cols(); ++j) {
          C(i,j) +=  t*z[k](j);
        }
      }
  }
}
```

5000 faces
Non-blocked: 10 cores- 22.2184 GFlops, 20 cores- 38.2374 GFlops
Blocked: 10 cores- 26.6679 GFlops, 20 cores- 35.7805 GFlops

Improvement in the case of 50000 faces is more noticeable.
50000 faces
Non-blocked: 10 cores- 19.6466 GFlops, 20 cores- 42.2224 GFlops
Blocked: 10 cores- 26.7273 GFlops, 20 cores- 58.1565 GFlops

DGEMM (Extra Credit)

```
auto gen_covariance(const std::vector<Vector>& z) {
  size_t sz = z[0].num_rows();
  Matrix F(sz, z.size());
  Matrix C(sz, sz);
  // Replace me with call to cblas_dgemm
  for (size_t k = 0; k < z.size(); ++k) {
      for (size_t i = 0; i < C.num_rows(); ++i) {
       F(i, k) = z[k](i);     // F matrix construction
       //Ft(k,i) = z[k](i);
      }
    }
 double alpha = 1;
 double beta = 0;
  cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasTrans, sz, sz, z.size(), alpha,
&F(0,0), z.size(), &F(0,0), z.size(), beta, &C(0,0), sz);
```

* What single core performance do you get with eigenfaces_dgemm? How does your performance scale for 2, 5, 10, 20 cores? How does your performance scale with number of faces (500, 5k, 50k)? (You may even try 40 cores and 50k or more faces.)

Number of cores
5000 faces: 1 core- 84.2589 GFlops, 2 cores- 140.819 GFlops, 5 cores- 220.346 GFlops, 10 cores- 319.043 GFlops, 20 cores- 306.281 GFlops

---

Number of faces
1 core:
500 faces- 60.649 GFlops,
5k faces- 82.003 GFlops,
50k faces- 12.183 GFlops

---

5 cores:
500 faces- 66.5829 GFlops,
5k faces- 213.436 GFlops,
50k faces- 268.432 GFlops

---

10 cores:
500 faces- 145.848 GFlops,
5k faces- 324.107 GFlops,
50k faces- 375.115 GFlops

---

20 cores:
500 faces- 95.7129 GFlops,
5k faces- 325.831 GFlops,
50k faces- 436.298 GFlops,

---

40 cores:
500 faces- 75.625 GFlops,
5k faces- 101.755 GFlops,
50k faces- 498.018 GFlops

MPI

Covariance computed using cblas_dgemm

```
// Replace me
auto gen_covariance(const std::vector<Vector>& z) {
  size_t sz = z[0].num_rows();
    Matrix F(sz, z.size());
    Matrix C(sz, sz);

    for (size_t k = 0; k < z.size(); ++k) {
        for (size_t i = 0; i < C.num_rows(); ++i) {
            F(i, k) = z[k](i);

        }
    }
    double alpha = 1;
    double beta = 0;
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasTrans, sz, sz, z.size(),
alpha, &F(0,0), z.size(), &F(0,0), z.size(), beta, &C(0,0), sz);
  return C;
}
```

```
    if (mysize > 1) {
      size_t count = sendcounts[1];
      for (size_t rank = 1; rank < mysize; ++rank) {
        for (size_t i = 0; i < sendcounts[rank]; ++i) {
    // Send z[count]
    // Write me
          MPI::COMM_WORLD.Send(&z[count](0), height * width, MPI::DOUBLE, rank,
321);
          ++count;
        }
      }
    }
```

```
    // Everyone else receive images from rank 0
    for (size_t i = 0; i < sendcounts[myrank]; ++i) {

      // Receive myimg_data[i]
      // Write me
      MPI::COMM_WORLD.Recv(&myimg_data[i](0), height * width, MPI::DOUBLE, 0 ,
321);
    }
```

```
  // Write me
  // Add up all the local_sums to global sum
  // All processes need global sum
  Vector sum(height * width);
  MPI::COMM_WORLD.Allreduce(&local_sum(0), &sum(0), height*width , MPI::DOUBLE,
MPI::SUM);
```

```
  // Write me
  // Add up all the local_Cs to C
  // Only rank 0 needs C
  Matrix C(local_C.num_rows(), local_C.num_cols());
  MPI::COMM_WORLD.Reduce(&local_C(0,0), &C(0,0), C.num_rows() * C.num_cols(),
MPI::DOUBLE, MPI::SUM,0);
```

* How does the performance of your eigenfaces_mpi.exe scale for 1, 2, 4, 8 nodes? (With 10 cores on each node.)

5000 images
1 node: 95.8627 GFlops; 2 nodes: 146.196 GFlops; 4 nodes: 273.692 GFlops
8 nodes: 631.508 GFlops

50000 images
1 node: 378.827 GFlops; 2 nodes: 550.866 GFlops; 4 nodes: 861.551 GFlops
8 nodes: 1431.22 GFlops

Max
---

* What configuration of nodes and cores per node gives you the very best performance? (And what was your best performance?)

For 50000 images:
The best performance is given by 4 nodes and 20 cores (3954.88 GFlops/s)
2 nodes and 40 cores has similar performance (3700 GFlops/s), for MED image sizes.
For SMALL image sizes, again 40 cores and 2 nodes (2561.17 GFlops/s) and 20 cores and 4 nodes (2314.45 GFlops/s) had similar performance and best among all the configurations.
In general performance increase with the number of cores used. The number of nodes led to marginal improvements.