Contents

Matrix Problems

- Matrices are very important for certain types of computationally-intensive problems
- Mostly problems related to the simulation of physical systems
  - Wind tunnel simulation
  - Nuclear weapons simulation
  - Weather forecasting
  - Flood mapping
  - etc
- Matrices are commonly used to store the coefficients of variables in large sets of simultaneous equations
- Programs contain many operations on large matrices
  - Matrix multiplication
  - Matrix inversion
  - Gaussian elimination
  - etc
- Data sets for matrix problems are typically large
  - Huge matrices with billions of elements
- Simulation of physical systems can be done at different levels of detail
  - Usually want as much detail as possible
  - Amount of detail is limited by processing speed/memory
  - Faster computation allows more details

- Many matrix problems can be parallelised extremely well
  - Especially matrix multiplication
  - But also others
- Historically, parallel computing was almost entirely focused on executing large matrix problems in parallel

Matrix by Vector Multiplication

```
y = x * A
```

- `x`, `y` are vectors and `A` is a matrix

```
for(i = 0; i < N; i++) {
    y[i] = 0.0;
    for(j = 0; j < N; j++) {
        y[i] += x[j] * A[i][j];
    }
}
```

Interchanging the Loops

```
for(j = 0; j < N; j++) {
    y[i] = 0.0;
    for(i = 0; i < N; i++) {
        for(j = 0; j < N; j++) {
            y[j] += x[i] * A[j][i];
        }
    }
}
```

Original with Caching

```
// cache y[i] in a local variable
for(i = 0; i < N; i++) {
    sum = 0.0;
    for(j = 0; j < N; j++) {
        sum += x[j] * A[i][j];
    }
    y[i] = sum;
}
```

Matrix by Matrix Multiplication

```
C = A * B
```

- A, B and C are matrices

Locality

- A common pattern arises in matrix operations
  - Simple way to write the code is to scan over all the rows (or columns) with a single loop
  - But this can result in poor reuse of values in the arrays
- Array elements are pulled into the cache when first used
- If those elements will be reused by the algorithm, we want to reuse them from the cache
  - Want to avoid reloading data from memory
- Rewriting the code to reuse data usually involves
  - Reordering loop iterations or
  - Reordering data in memory

Matrix by Vector Multiplication

Halve number of times `x` must be loaded

```
for(i = 0; i < N; i+=2) {
    sum0 = 0.0;
    sum1 = 0.0;
    for(j = 0; j < N; j++) {
        sum0 += x[j] * A[i][j];
        sum1 += x[j] * A[i+1][j];
    }
    y[i] = sum0;
    y[i+1] = sum1;
}
```

Quarter number of times `x` must be loaded

```
for(i = 0; i < N; i+=4) {
    sum0 = 0.0; sum1 = 0.0; sum2 = 0.0; sum3 = 0.0;
    for(j = 0; j < N; j++) {
```

```
        sum0 += x[j] * A[i][j];
        sum1 += x[j] * A[i+1][j];
        sum2 += x[j] * A[i+2][j];
        sum3 += x[j] * A[i+3][j];
    }
    y[i] = sum0;
    y[i+1] = sum1;
    y[i+2] = sum2;
    y[i+3] = sum3;
}
```

Reduce loading of `x` to cache by factor of `k`

```
for(i = 0; i < N; i+=k) {
    for(tmp = 0; tmp < k; tmp++) {
        sum[tmp] = 0.0;
    }
    for(j = 0; j < N; j++) {
        for(tmp = 0; tmp < k; tmp++) {
            sum[tmp] += x[j] * A[i+tmp][j];
        }
    }
    for(tmp = 0; tmp < k; tmp++) {
        y[i+tmp] = sum[tmp];
    }
}
```

Make it work if `N` is not a multiple of `k`

```
for(i = 0; i < N; i+=k) {
    stop = (i+k < N) ? k : N-1;
    for(tmp = 0; tmp < stop; tmp++) {
        sum[tmp] = 0.0;
    }
    for(j = 0; j < N; j++) {
        for(tmp = 0; tmp < stop; tmp++) {
            sum[tmp] += x[j] * A[i+tmp][j];
        }
    }
    for(tmp = 0; tmp < stop; tmp++) {
        y[i+tmp] = sum[tmp];
    }
}
```

Improving Locality

- This technique can be used to reduce cache misses on `x` a lot
  - i.e. close to the minimum possible
- However, the inner loop that does the "blocking" of the `x` vector accesses different elements of the `y` vector
  - If we keep increasing `k` (the "blocking factor") we will eventually get a lot of cache misses on accesses to the `y` vector
- So how big should `k` be?
- The "optimal" value of `k` will be one that keeps as much as feasible of `x` in cache
  - Obvious answer is as much as will fit in cache
- However
  - Cache is also used by other data
  - Multiple levels of cache
  - With `k=4`, we already achieve 75% of possible benefit
- Best choice of `k` is usually difficult to find analytically
- In practice auto-tuners are often used
  - An auto-tuner is a program that is used to tune another program
  - Auto-tuner tries out different values of constants that affect performance
    * Compile, run, time, feedback, start again
  - Auto-tuners often use machine learning techniques
    * Huge search space of possible solutions

Matrix Multiplication

- Matrix multiplication (matmul) is similar to matrix vector multiplication
  - Multiply two matrices to get a result which is a matrix
- For `N` $\times$ `N` matrices, there will be $O(N^3)$ operations
  - Using the straightforward algorithm
  - More computationally internsive than matrix-vector multiplication
  - Large ratio of computation to memory ops

```
for(i = 0; i < N; i++) {
    for(j = 0; j < N; j++) {
        sum = 0;
        for(k = 0; k < N; k++) {
```

```
            sum += a[i][k] * b[k][j];
        }
        c[i][j] = sum;
    }
}
```