МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий

Кафедра параллельных вычислений

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

«Векторизация вычислений»

Студентки 2 курса, 24202 группы

Корсун Дарьи Андреевны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Кандидат технический наук
В.А. Перепелки

Новосибирск 2025

СОДЕРЖАНИЕ

ЦЕЛЬ

1. Изучение SIMD-расширений архитектуры x86/x86-64.
2. Изучение способов использования SIMD-расширений в программах на языке Си.
3. Получение навыков использования SIMD-расширений.

ЗАДАНИЕ

1. Написать три варианта программы, реализующей алгоритм из задания:
   -вариант без ручной векторизации,
   -вариант с ручной векторизацией
   -вариант с матричными операциями(BLAS).
2. Проверить правильность работы программ на нескольких небольших тестовых наборах входных данных.
3. Каждый вариант программы оптимизировать по скорости, насколько это возможно.
4. Сравнить время работы трех вариантов программы для N=2048, M=10.
5. Составить отчет по лабораторной работе..

ОПИСАНИЕ РАБОТЫ

Были написаны программы которые соответственно реализуют разложение матрицы через обычный подсчет, с использованием ручной векторизации (в данной работе использовались расширения GCC) и с использованием матричных операций из готовой оптимизированной библиотеки BLAS.

Таблица результатов для нескольких измерений, чтобы снизить влияние побочных фактором на подсчет результатов.

N	Без вектор.	Simd	Blas
1	104,20	42,55	0,48
2	105,19	42,33	0,47
3	104,91	42,44	0,49
4	105,67	43,15	0,45

Как можно видеть из результатов таблицы, готовая библиотека намного лучше справляется с оптимизацией матричных вычислений, хоть ручная векторизация и уменьшила время вычисления в 2 раза, это не сравнится с результатом полученным от вычислений с помощью Blas.

Ниже прикладываю код, используемый для запуска:

-Без векторизации

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void matmul(float *C, float *A, float *B, int N) {
  for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++) {
      float sum = 0.0f;
      for (int k = 0; k < N; k++) sum += A[i * N + k] * B[k * N
+ j];
      C[i * N + j] = sum;
    }
}

void matadd(float *C, float *A, float *B, int N) {
  for (int i = 0; i < N * N; i++) C[i] = A[i] + B[i];
}

void eye(float *I, int N) {
  for (int i = 0; i < N * N; i++) I[i] = 0.0f;
  for (int i = 0; i < N; i++) I[i * N + i] = 1.0f;
}

void transpose(float *At, float *A, int N) {
```

```
  for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++) At[j * N + i] = A[i * N + j];
}

float norm1(float *A, int N) {
  float max = 0.0f;
  for (int j = 0; j < N; j++) {
    float sum = 0.0f;
    for (int i = 0; i < N; i++) sum += fabsf(A[i * N + j]);
    if (sum > max) max = sum;
  }
  return max;
}

float norm_inf(float *A, int N) {
  float max = 0.0f;
  for (int i = 0; i < N; i++) {
    float sum = 0.0f;
    for (int j = 0; j < N; j++) sum += fabsf(A[i * N + j]);
    if (sum > max) max = sum;
  }
  return max;
}

int main() {
  int N = 2048, M = 10;

  float *A = (float *)malloc(N * N * sizeof(float));
  float *At = (float *)malloc(N * N * sizeof(float));
  float *B = (float *)malloc(N * N * sizeof(float));
  float *R = (float *)malloc(N * N * sizeof(float));
  float *I = (float *)malloc(N * N * sizeof(float));
  float *tmp = (float *)malloc(N * N * sizeof(float));
  float *invA = (float *)malloc(N * N * sizeof(float));
  float *Rpow = (float *)malloc(N * N * sizeof(float));

   for (int i = 0; i < N * N; i++) {
    A[i] = (float)rand() / RAND_MAX;
  }

  clock_t start_time = clock();

  transpose(At, A, N);

  float n1 = norm1(A, N);
  float ninf = norm_inf(A, N);
  for (int i = 0; i < N * N; i++) B[i] = At[i] / (n1 * ninf);

  matmul(tmp, B, A, N);
  eye(I, N);
  for (int i = 0; i < N * N; i++) R[i] = I[i] - tmp[i];

  eye(invA, N);
  for (int i = 0; i < N * N; i++) Rpow[i] = R[i];
  for (int m = 1; m < M; m++) {
    matadd(invA, invA, Rpow, N);
    matmul(tmp, Rpow, R, N);
    for (int i = 0; i < N * N; i++) Rpow[i] = tmp[i];
  }
```

```
  matmul(tmp, B, invA, N);

  clock_t end_time = clock();
  double execution_time = (double)(end_time - start_time) /
CLOCKS_PER_SEC;

  printf("Plain C execution time for N=%d: %.4f seconds\n", N,
execution_time);


  free(A);
  free(At);
  free(B);
  free(R);
  free(I);
  free(tmp);
  free(invA);
  free(Rpow);

  return 0;
}
```

-Simd

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

typedef float v4sf __attribute__((vector_size(16)));

void vec_add(float *C, float *A, float *B, int N) {
  for (int i = 0; i < N * N; i += 4) {
    v4sf a = *((v4sf *)&A[i]);
    v4sf b = *((v4sf *)&B[i]);
    *((v4sf *)&C[i]) = a + b;
  }
}

void vec_sub(float *C, float *A, float *B, int N) {
  for (int i = 0; i < N * N; i += 4) {
    v4sf a = *((v4sf *)&A[i]);
    v4sf b = *((v4sf *)&B[i]);
    *((v4sf *)&C[i]) = a - b;
  }
}

void vec_mul(float *C, float *A, float *B, int N) {
  for (int i = 0; i < N * N; i += 4) {
    v4sf a = *((v4sf *)&A[i]);
    v4sf b = *((v4sf *)&B[i]);
    *((v4sf *)&C[i]) = a * b;
  }
}

void vec_div_scalar(float *C, float *A, float scalar, int N) {
  v4sf s = {scalar, scalar, scalar, scalar};
  for (int i = 0; i < N * N; i += 4) {
    v4sf a = *((v4sf *)&A[i]);
    *((v4sf *)&C[i]) = a / s;
```

```
  }
}

void matmul_simd(float *C, float *A, float *B, int N) {
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
      float sum = 0.0f;

      for (int k = 0; k < N; k += 4) {
        v4sf a_vec, b_vec, mul_vec;

        if (k + 3 < N) {
          a_vec = *((v4sf *)&A[i * N + k]);
          b_vec = *((v4sf *)&B[k * N + j]);

          mul_vec = a_vec * b_vec;
          sum += mul_vec[0] + mul_vec[1] + mul_vec[2] +
mul_vec[3];
        } else {
          for (int kk = k; kk < N; kk++) {
            sum += A[i * N + kk] * B[kk * N + j];
          }
        }
      }
      C[i * N + j] = sum;
    }
  }
}

void transpose(float *At, float *A, int N) {
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
      At[j * N + i] = A[i * N + j];
    }
  }
}

void eye(float *I, int N) {
  for (int i = 0; i < N * N; i++) I[i] = 0.0f;
  for (int i = 0; i < N; i++) I[i * N + i] = 1.0f;
}

float norm1(float *A, int N) {
  float max = 0.0f;
  for (int j = 0; j < N; j++) {
    float sum = 0.0f;

    int i;
    for (i = 0; i <= N - 4; i += 4) {
      v4sf a_vec = *((v4sf *)&A[i * N + j]);
      v4sf abs_vec;

      for (int k = 0; k < 4; k++) {
        abs_vec[k] = fabsf(a_vec[k]);
      }
      sum += abs_vec[0] + abs_vec[1] + abs_vec[2] + abs_vec[3];
    }
    for (; i < N; i++) {
      sum += fabsf(A[i * N + j]);
    }
```

```
    if (sum > max) max = sum;
  }
  return max;
}

float norm_inf(float *A, int N) {
  float max = 0.0f;
  for (int i = 0; i < N; i++) {
    float sum = 0.0f;
    int j;
    for (j = 0; j <= N - 4; j += 4) {
      v4sf a_vec = *((v4sf *)&A[i * N + j]);
      v4sf abs_vec;
      for (int k = 0; k < 4; k++) {
        abs_vec[k] = fabsf(a_vec[k]);
      }
      sum += abs_vec[0] + abs_vec[1] + abs_vec[2] + abs_vec[3];
    }

    for (; j < N; j++) {
      sum += fabsf(A[i * N + j]);
    }
    if (sum > max) max = sum;
  }
  return max;
}

int main() {
  int N = 2048, M = 10;


  float *A = (float *)malloc(N * N * sizeof(float));
  float *At = (float *)malloc(N * N * sizeof(float));
  float *B = (float *)malloc(N * N * sizeof(float));
  float *R = (float *)malloc(N * N * sizeof(float));
  float *I = (float *)malloc(N * N * sizeof(float));
  float *tmp = (float *)malloc(N * N * sizeof(float));
  float *invA = (float *)malloc(N * N * sizeof(float));
  float *Rpow = (float *)malloc(N * N * sizeof(float));


  for (int i = 0; i < N * N; i++) {
    A[i] = (float)rand() / RAND_MAX;
  }

  clock_t start_time = clock();

  transpose(At, A, N);

  float n1 = norm1(A, N);
  float ninf = norm_inf(A, N);
  vec_div_scalar(B, At, n1 * ninf, N * N);

  matmul_simd(tmp, B, A, N);
  eye(I, N);
  vec_sub(R, I, tmp, N * N);

  eye(invA, N);
  for (int i = 0; i < N * N; i++) Rpow[i] = R[i];
```

```
  for (int m = 1; m < M; m++) {
    vec_add(invA, invA, Rpow, N * N);
    matmul_simd(tmp, Rpow, R, N);
    for (int i = 0; i < N * N; i++) Rpow[i] = tmp[i];
  }

  matmul_simd(tmp, B, invA, N);

  clock_t end_time = clock();
  double execution_time = (double)(end_time - start_time) /
CLOCKS_PER_SEC;

  printf("SIMD execution time for N=%d: %.4f seconds\n", N,
execution_time);
  free(A);
  free(At);
  free(B);
  free(R);
  free(I);
  free(tmp);
  free(invA);
  free(Rpow);

  return 0;
}
```

-Blas

```
#include <cblas.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
  int N = 2048, M = 10;

  float *A = (float *)malloc(N * N * sizeof(float));
  float *At = (float *)malloc(N * N * sizeof(float));
  float *B = (float *)malloc(N * N * sizeof(float));
  float *I_mat = (float *)malloc(N * N * sizeof(float));
  float *R = (float *)malloc(N * N * sizeof(float));
  float *tmp = (float *)malloc(N * N * sizeof(float));
  float *invA = (float *)malloc(N * N * sizeof(float));

  for (int i = 0; i < N * N; i++) {
    A[i] = (float)rand() / RAND_MAX;
  }

  clock_t start_time = clock();

  for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++) At[j * N + i] = A[i * N + j];

  float n1 = 0, ninf = 0;
  for (int j = 0; j < N; j++) {
    float sum = 0;
    for (int i = 0; i < N; i++) sum += fabsf(A[i * N + j]);
    if (sum > n1) n1 = sum;
  }
```

```
  for (int i = 0; i < N; i++) {
    float sum = 0;
    for (int j = 0; j < N; j++) sum += fabsf(A[i * N + j]);
    if (sum > ninf) ninf = sum;
  }

  for (int i = 0; i < N * N; i++) B[i] = At[i] / (n1 * ninf);

  for (int i = 0; i < N * N; i++) I_mat[i] = 0.0f;
  for (int i = 0; i < N; i++) I_mat[i * N + i] = 1.0f;

  cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N,
N, -1.0f, B, N,
              A, N, 1.0f, I_mat, N);
  for (int i = 0; i < N * N; i++) R[i] = I_mat[i];

  for (int i = 0; i < N * N; i++) invA[i] = I_mat[i];
  cblas_scopy(N * N, R, 1, tmp, 1);
  for (int m = 1; m < M; m++) {
    cblas_saxpy(N * N, 1.0f, tmp, 1, invA, 1);
    cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N,
N, N, 1.0f, tmp,
                N, R, N, 0.0f, I_mat, N);
    cblas_scopy(N * N, I_mat, 1, tmp, 1);
  }

  cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N,
N, 1.0f, B, N,
              invA, N, 0.0f, I_mat, N);

  clock_t end_time = clock();
  double execution_time = (double)(end_time - start_time) /
CLOCKS_PER_SEC;

  printf("OpenBLAS execution time for N=%d: %.4f seconds\n", N,
execution_time);

  free(A);
  free(At);
  free(B);
  free(I_mat);
  free(R);
  free(tmp);
  free(invA);

  return 0;
}
```

ЗАКЛЮЧЕНИЕ

Ручная векторизация хорошо справляется с задачей не типизированной готовыми библиотеками, то есть дает больше свободы в применении для написаний кода соответственно широкий диапазон для ускорения различного рода кода, в то время как готовая конкретная библиотека помогает получить очень хорошее ускорение для конкретной залачи (подсчет матриц, как в данной лабораторной).