CS601
Software Development for Scientific Computing
Assignment 1

Cebajel Tanan - 210010055
Balaji NK - 210010008
Aditya Sujeet Zanjurne - 210010001

15th September, 2023

Approx Reading time: 5min

Contribution break-up

Cebajel Tanan

- Makefile
- Problem 1(b), 2 and 3

Balaji NK

- Readme file
- Problem 1(a)
- Report: tables

Aditya Sujeet Zanjurne

- Problem 1(c)
- Report: images, described observations

Problem Statements

Problem 1: Matrix Multiplication

The starter file for this part of the assignment contains `matmul.cpp`, which implements a C++ program for computing the product of two matrices as follows: $C = C + A \cdot B$, where C, A, and B are square matrices of size N. The product $A \cdot B$ is computed and updated to matrix C, which is zero-initialized. A simple matrix multiplication consists of 3 nested loops (referred to as the `ijk` loop) shown in the below pseudocode:

for i = 0 to N - 1
    for j = 0 to N - 1
        for k = 0 to N - 1
            $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$

a_{xy}, b_{xy}, and c_{xy} are elements of matrices A, B, and C, respectively, with x and y being suitable indices in the range $[0, N-1]$.

(a) Implement Different Versions of Matrix Multiplication

Implement different versions of the matrix multiplication function corresponding to every possible loop ordering (`ijk`, `ikj`, `jik`, etc.). Tabulate the execution times and throughputs in your report. Edit the `Makefile` to build separate targets for each of the loop orderings and execute them with a matrix size of 2048×2048. You may edit the file provided as per your wish. However, you must include conditional compilation for compiling different versions of matmul implementations.

(b) Matrix Multiplication with Different Optimization Levels

Consider the `ijk` implementation of matmul. Build this version with the following optimization levels: (a) no optimizations, (b) `-O1`, (c) `-O2`, (d) `-O3`, (e) `-O4`. Tabulate the execution times and throughputs for different optimization levels. Edit the `Makefile` to build multiple targets with names `matmul_Ox`, where `x` is the optimization level. For each of the targets built, the `make` command should also execute them with a matrix size of 2048×2048. You need not edit the `matmul.cpp` file provided for this part.

(c) Matrix Multiplication Using BLAS Library

Read about and use library functions from the BLAS library for performing matrix multiplication. Implement two versions: (a) using the available function for performing dot-product, (b) using `sgemm`. Tabulate the execution times and throughputs. Discuss your observations.

Problem 2: Matrix-Vector Multiplication with SSE3 Intrinsics

The starter files for this part of the assignment are `matvec.cpp`, `timeutils.cpp`, and `timeutils.h`. Read about and use the function calls / intrinsics for SSE3 (128-bit vector register extension) on x86 architecture. Implement matrix-vector product using intrinsics. Your implementation in the `matvec_intrinsics()` function should make calls to functions that look like e.g., `_mm_load_ps()` etc. The resulting implementation would be equivalent to unrolling the inner loop 4 times. Edit the `matvec.cpp` file to implement this part of the assignment. Discuss your observations. Also, edit the `Makefile` so that after building the required target, you execute the target.

Problem 3(Bonus): Matrix-Matrix Multiplication with SSE3 Intrinsics

Implement a version of `matmul_intrinsics` for executing matrix multiplication using advanced intrinsics such as AVX or AVX-512.

Execution times and throughputs

Problem 1(a): matmul_schedule

loop orderings

Loop order	time(s)	throughput(flops)
ijk	74.7395	2.29863e+08
ikj	27.5557	6.2346e+08
kij	27.2151	6.31262e+08
kji	69.8501	2.45954e+08
jik	69.4165	2.4749e+08
jki	78.3502	2.1927e+08

```
[cs601user10@hip cs601pa1-Cebajel]$ make matmul_schedule
./bin/matmul_ijk 2048
elapsed seconds: 74.7395 s
throughput: 2.29863e+08 flops
e_sum: 0e+00
Result OK
echo ""

./bin/matmul_ikj 2048
elapsed seconds: 27.5557 s
throughput: 6.2346e+08 flops
e_sum: 0e+00
Result OK
echo ""

./bin/matmul_kij 2048
elapsed seconds: 27.2151 s
throughput: 6.31262e+08 flops
e_sum: 0e+00
Result OK
echo ""

./bin/matmul_kji 2048
elapsed seconds: 69.8501 s
throughput: 2.45954e+08 flops
e_sum: 0e+00
Result OK
echo ""

./bin/matmul_jik 2048
elapsed seconds: 69.4165 s
throughput: 2.4749e+08 flops
e_sum: 0e+00
Result OK
echo ""

./bin/matmul_jki 2048
elapsed seconds: 78.3502 s
throughput: 2.1927e+08 flops
e_sum: 0e+00
Result OK
echo ""
```

Problem 1(b): matmul_optlevel

optimizations

Optimization	time(s)	throughput(flops)
no optimization	71.0072	2.41945e+08
o1	24.38	7.046716e+08
o2	24.2579	7.08218e+08
o3	24.3319	7.06063e+08
o4	24.3318	7.06065e+08

```
[cs601user10@hip cs601pa1-Cebajel]$ make matmul_optlevel
./bin/matmul_o0 2048
elapsed seconds: 71.0072 s
throughput: 2.41945e+08 flops
e_sum: 0e+00
Result OK
echo ""

./bin/matmul_o1 2048
elapsed seconds: 24.38 s
throughput: 7.04671e+08 flops
e_sum: 0e+00
Result OK
echo ""

./bin/matmul_o2 2048
elapsed seconds: 24.2579 s
throughput: 7.08218e+08 flops
e_sum: 0e+00
Result OK
echo ""

./bin/matmul_o3 2048
elapsed seconds: 24.3319 s
throughput: 7.06063e+08 flops
e_sum: 0e+00
Result OK
echo ""

./bin/matmul_o4 2048
elapsed seconds: 24.3318 s
throughput: 7.06065e+08 flops
e_sum: 0e+00
Result OK
echo ""
```

Problem 1(c): matmul_blas

blas functions

Function	time(s)	throughput(flops)
`cblas_sdot()`	1.36472	1.25885e+10
`cblas_sgemm()`	0.0241711	7.10761e+11

```
[cs601user10@hip cs601pa1-Cebajel]$ make matmul_blas
./bin/matmul_a 2048
Using cblas_sdot function:
elapsed seconds: 1.36472 s
throughput: 1.25885e+10 flops
e_sum: 1e+03
Result MISMATCH
echo ""

./bin/matmul_b 2048
Using sgemm function:
elapsed seconds: 0.0241711 s
throughput: 7.10761e+11 flops
e_sum: 1e+03
Result MISMATCH
echo ""
```

Problem 2: matvec_intrinsics

optimized matvec

Operation	time(s)	throughput(flops)
Matrix-Vector Multiplication	0.02	2.7128e+10

Matvec reference code completed in 0.29 s
Speedup: 14.72

```
obj/matvec
Matvec using intrinsics completed in 0.02 s
Throughtput: 2.7128e+10 flops
Matvec reference code completed in 0.29 s
Speedup: 14.72
e_sum: 0e+00
Result OK
echo ""
```

- We have implemented matvec using two loops out of which the inner one is implemented using **loop unrolling**.
- The inner loop reads 2 sets **16 single precision** floating point numbers in one iteration.
- These are stored in two vectors(`__m512 registor1, __m512 registor2`) and multiplied using `_mm512_mul_ps()` function. Note that we are doing 16 multiplications at a time.
- Thereafter, the result is added to the `result` vector.
- `_mm512_reduce_add_ps()` function adds up all the values stored in the `result` vector to give a scalar which is then added to the correct index in `vec_c`. Thus, the dot product is done which gives the corresponding element of `vec_c`.

Problem 3: matmul_intrinsics

optimized matmul

Operation	time(s)	throughput(flops)
Matrix-Matrix Multiplication	0.03	5.5276e+11

Matmul reference code completed in 17.78 s
Speedup: 571.98

```
obj/matvec_matmul
Matmul using intrinsics completed in 0.03 s
Throughput: 5.5276e+11 flops
Matmul reference code completed in 17.78 s
Speedup: 571.98
e_sum: 0e+00
Result OK
echo ""
```

- The innermost loop works the same as the one in **matvec**.
- After this the `_mm512_reduce_add_ps()` function adds up all the values stored in `result` to give a scalar which is then added to the correct index in `mat_c`.

Conclusion

We have successfully completed Assignment 1. In this assignment, we learnt the various methods of performing Matrix-Matrix Multiplication(loop ordering, optimizations, BLAS library, SSE3 intrinsics). We also learnt how to do Matrix-Vector Multiplication using SSE3 intrinsics.
According to the results we have obtained, we can conclude that the most effective method of Matrix-Matrix Multiplication is by using the `cblas_sgemm()` function of BLAS library as it gives us the highest throughput among all methods.
We also see how effective SSE3 intrinsics are for Matrix-Vector Multiplication and Matrix-Matrix Multiplication as they are $\sim$15 times and $\sim$ 572 times faster than normal `ijk` loop implementation respectively.