MATRIX MULTIPLICATION

STEP BY STEP SPEED UP

NISANTH M P, 2023 JAN 27

The Definition

If $\mathbf{A}$ is an $m \times n$ matrix and $\mathbf{B}$ is an $n \times p$ matrix,

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

the *matrix product* $\mathbf{C} = \mathbf{AB}$ (denoted without multiplication signs or dots) is defined to be the $m \times p$ matrix

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

such that

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj},$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, p$.

Reference

Step 01: Definition in Code

```
for(int i = 0; i < n; ++i)
    for(int j = 0; j < n; ++j)
        for(int k = 0; k < n; ++k)
        {
            C[i][j] += A[i][k] * B[k][j];
        }
```

n = 2048, GCC 13.0, No Optimisation: ~50 seconds

n = 2048, GCC 13.0, O3 Optimisation: ~17 seconds
(ie. ~3x improvement with compiler optimisations - no change in code)

System Details

Step 02: Swap Indices

```
for(int i = 0; i < n; ++i)
    for(int k = 0; k < n; ++k)
        for(int j = 0; j < n; ++j)
        {
            C[i][j] += A[i][k] * B[k][j];
        }
```

Step 01: n = 2048, GCC 13.0, O3 Optimisation: ~17 seconds

Step 02: n = 2048, GCC 13.0, O3: ~2.32 seconds
(ie. ~7x improvement with single line of change in code)

Reason?

System Details

Step 03: Loop Unroll k

```
for(int i = 0; i < n; ++i)
    for(int k = 0; k < n; ++k)
        for(int j = 0; j < n; j += 16)
        {
            C(i,j) += A(i,k) * B(k,j);
            C(i,j+1) += A(i,k) * B(k,j+1);
            C(i,j+2) += A(i,k) * B(k,j+2);
            C(i,j+3) += A(i,k) * B(k,j+3);
            C(i,j+4) += A(i,k) * B(k,j+4);
            C(i,j+5) += A(i,k) * B(k,j+5);
            C(i,j+6) += A(i,k) * B(k,j+6);
            C(i,j+7) += A(i,k) * B(k,j+7);
            C(i,j+8) += A(i,k) * B(k,j+8);
            C(i,j+9) += A(i,k) * B(k,j+9);
            C(i,j+10) += A(i,k) * B(k,j+10);
            C(i,j+11) += A(i,k) * B(k,j+11);
            C(i,j+12) += A(i,k) * B(k,j+12);
            C(i,j+13) += A(i,k) * B(k,j+13);
            C(i,j+14) += A(i,k) * B(k,j+14);
            C(i,j+15) += A(i,k) * B(k,j+15);
        }
```

Step 02: n = 2048, GCC 13.0, O3:
~2.32 seconds

Step 03: n = 2048, GCC 13.0, O3:
~XXXX seconds

System Details

Step 03: Loop Unroll j

```
for(int i = 0; i < n; ++i)
    for(int k = 0; k < n; ++k)
        for(int j = 0; j < n; j += 16)
        {
            C(i,j) += A(i,k) * B(k,j);
            C(i,j+1) += A(i,k) * B(k,j+1);
            C(i,j+2) += A(i,k) * B(k,j+2);
            C(i,j+3) += A(i,k) * B(k,j+3);
            C(i,j+4) += A(i,k) * B(k,j+4);
            C(i,j+5) += A(i,k) * B(k,j+5);
            C(i,j+6) += A(i,k) * B(k,j+6);
            C(i,j+7) += A(i,k) * B(k,j+7);
            C(i,j+8) += A(i,k) * B(k,j+8);
            C(i,j+9) += A(i,k) * B(k,j+9);
            C(i,j+10) += A(i,k) * B(k,j+10);
            C(i,j+11) += A(i,k) * B(k,j+11);
            C(i,j+12) += A(i,k) * B(k,j+12);
            C(i,j+13) += A(i,k) * B(k,j+13);
            C(i,j+14) += A(i,k) * B(k,j+14);
            C(i,j+15) += A(i,k) * B(k,j+15);
        }
```

Step 02: n = 2048, GCC 13.0, O3: ~2.32 seconds

Step 03: n = 2048, GCC 13.0, O3: ~2.30 seconds

NO CHANGE!!

Reason?

System Details

Step 04: Loop Unroll i

```
for(int i = 0; i < n; i += 4)
    for(int k = 0; k < n; ++k)
        for(int j = 0; j < n; j += 16)
        {
            C(i,j) += A(i,k) * B(k,j);
            C(i,j+1) += A(i,k) * B(k,j+1);
            ;
            ;
            C(i,j+15) += A(i,k) * B(k,j+15);

            C(i+1,j) += A(i+1,k) * B(k,j);
            C(i+1,j+1) += A(i+1,k) * B(k,j+1);
            ;
            ;
            C(i+1,j+15) += A(i+1,k) * B(k,j+15);

            C(i+2,j) += A(i+2,k) * B(k,j);
            C(i+2,j+1) += A(i+2,k) * B(k,j+1);
            ;
            ;
            C(i+2,j+15) += A(i+2,k) * B(k,j+15);

            C(i+3,j) += A(i+3,k) * B(k,j);
            C(i+3,j+1) += A(i+3,k) * B(k,j+1);
            ;
            ;
            C(i+3,j+15) += A(i+3,k) * B(k,j+15);
        }
```

Step 03: n = 2048, GCC 13.0, O3:
~2.30 seconds

Step 04: n = 2048, GCC 13.0, O3:
~1.20 seconds

~2x improvement

Reason?

System Details

Step 05: Blocking

```
int mb = 128, nb = 128;
for (int nk = 0; nk < n; nk += nb)
    for (int mi = 0; mi < n; mi += mb) {
        double *a = &AS[mi][nk];
        double *b = &BS[nk][0];
        double *c = &CS[mi][0];

        // Matrix multiplication
        for(int i = 0; i < mb; i += 4)
            for(int k = 0; k < nb; ++k)
                for(int j = 0; j < n; j += 16)
                {
                    C(i,j) += A(i,k) * B(k,j);
                    ;
                    C(i,j+15) += A(i,k) * B(k,j+15);

                    C(i+1,j) += A(i+1,k) * B(k,j);
                    ;
                    C(i+1,j+15) += A(i+1,k) * B(k,j+15);

                    C(i+2,j) += A(i+2,k) * B(k,j);
                    ;
                    C(i+2,j+15) += A(i+2,k) * B(k,j+15);

                    C(i+3,j) += A(i+3,k) * B(k,j);
                    ;
                    C(i+3,j+15) += A(i+3,k) * B(k,j+15);
                }
    }
```

Step 04: n = 2048, GCC 13.0, O3:
~1.20 seconds

Step 05: n = 2048, GCC 13.0, O3:
~0.70 seconds

~2x improvement

Reason?

System Details

Step 06: Multithreading

```
int mb = 128, nb = 128;
#pragma omp parallel for
for (int nk = 0; nk < n; nk += nb)
    for (int mi = 0; mi < n; mi += mb) {
        double *a = &AS[mi][nk];
        double *b = &BS[nk][0];
        double *c = &CS[mi][0];

        // Matrix multiplication
        for(int i = 0; i < mb; i += 4)
            for(int k = 0; k < nb; ++k)
                for(int j = 0; j < n; j += 16)
                {
                    C(i,j) += A(i,k) * B(k,j);
                    ;
                    C(i,j+15) += A(i,k) * B(k,j+15);

                    C(i+1,j) += A(i+1,k) * B(k,j);
                    ;
                    C(i+1,j+15) += A(i+1,k) * B(k,j+15);

                    C(i+2,j) += A(i+2,k) * B(k,j);
                    ;
                    C(i+2,j+15) += A(i+2,k) * B(k,j+15);

                    C(i+3,j) += A(i+3,k) * B(k,j);
                    ;
                    C(i+3,j+15) += A(i+3,k) * B(k,j+15);
                }
    }
```

Step 05: n = 2048, GCC 13.0, O3:
~0.7 seconds

Step 06: n = 2048, GCC 13.0, O3:
~0.18 seconds

~3.5x improvement

System Details

System Details

system: IBM,9080-HEX
processor: PowerPC,POWER10
CPU clock: 3450MHz
CPU(s): 384
L1d cache: 32KB
L1i cache: 48KB
L2 cache: 1024KB
L3 cache: 4096KB
memory: 477GiB System memory
OS: Linux

Back to step: 01 02 03 04 05 06

Reasons

Step 01: Increased cache/register hits and xxxx

Back to step: 02

Reasons

Step 02: Increased cache/register hits and compiler optimised SIMD

Step 03: Compiler already did this ^ loop unrolling and SIMD optimisation

Back to step: 03

Reasons

Step 02: Increased cache/register hits and compiler optimised SIMD
Step 03: Compiler already did this ^ loop unrolling and SIMD optimisation
Step 04: Increased cache/register hits

Back to step: 04

Reasons

Step 02: Increased cache/register hits and compiler optimised SIMD
Step 03: Compiler already did this ^ loop unrolling and SIMD optimisation
Step 04: Increased cache/register hits
Step 05: Increased cache hits due to smaller blocks fitting in L2 cache

Back to step: 05

More Things to Try

07: Hand assembly optimisations to include special vector instructions etc - specific for ISAs

08: Experimenting with different loop unrolling factors - depends on register sizes and register file sizes

09: Experimenting with different block sizes - depends on L2 and L3 cache sizes

10: Packing blocks of matrices A & B - to increase cache hits

11: Experimenting with different thread nesting strategies to maximise parallelism - core scaling

12: Experimenting with different thread pinning strategies to optimise for NUMA scaling

References

Have borrowed some ideas from BLIS' GEMM speed up wiki - https://github.com/flame/how-to-optimize-gemm

Code and other documents available at: https://github.com/nisanthmp/matmul-speedup

Q&A

Thank You