中国矿业大学(北京)
China University of Mining & Technology, Beijing

Matrix Multiplication Acceleration

Implemented By Strassen Algorithm and Intel(R) Math Kernel Library

组员: 仇琨元 徐嘉睿 肖锐卓 吴雨飞

Insitute of EEE
SUSTC

December 19, 2020

Contents

Necessity of Matrix Multiplication Acceleration

Contents

Necessity of Matrix Multiplication Acceleration

The multiplication of two matrices is one of the most basic operations of linear algebra and scientific computing.

Modern signal processing, artificial intelligence and computer vision are all based on the fast and accurate algorithm of matrix multiplication, LU/QR/SVD decomposition and many other operations.

Comparison of the time costs of computing the FF

CPU	Clock Frequency	DFT	FFT
1941	60 Hz	152.3 y	271.4 d
1971 (4004)	108KHz	30.8 d	3.6 h
1978 (8086)	10MHz	8.0 h	2.3 min
1982 (80286)	20MHz	4.0 h	1.2min
1985 (80386)	33MHz	2.4h	42.6s
1989 (80486)	100MHz	48.0min	14.1s
1995 (Pentium)	200MHz	24.0min	7.0s
1999 (Pentium Ⅲ)	450MHz	10.7min	3.1s
2000 (Pentium 4)	1.4GHz	3.4min	1.0s
2001 (Pentium 4)	2GHz	2.4min	0.7s

Contents

Brute-Force Algorithm

The Strassen Multiplication uses divide-conquer to reduce the time complexity of MM operations. Normal MM uses 3 nested loops to perform the vector dotting and traversal of the rows and columns of the 2 operands:

```
STANDARD-MATRIX-MULTIPLY (A,B):
let C be a new m*n matrix
  for i <- 1 to m
    for j <- 1 to n
      C[i,j] = 0
        for k = 1 to p
            C[i,j] += A[i,k]*B[k,j]
  return C
```

Brute-Force Algorithm

From the pseudocode above, let MUL, ADD and READ refers the *assembly* commands of the computer. Each nested loop multiplies the time complexity by its iteration number:

$$\begin{aligned} T(\text{Naive}) &= m \cdot n \cdot p \cdot (T(\text{MUL} + \text{ADD} + 2\text{READ})) \\ &= \Theta(mnp) \\ m = n = p \Rightarrow T(\text{Naive}) &= \Theta(n^3) \end{aligned} \tag{1}$$

Strassen Algorithm

Pseudocode of the Strassen algorithm:

```
STRASSEN (MatrixA,MatrixB)
    N=MatrixA.rows
  Let MatrixResult be a new N N matrix
  if N==1
    MatrixResult=MatrixA*MatrixB
  else
     // DIVIDE: partitioning input Matrices into 4 submatrices each
       for i <- 0 to N/2
           for j <- 0 to N/2
              A11[i][j] <- MatrixA[i][j]
              A12[i][j] <- MatrixA[i][j + N/2]
              A21[i][j] <- MatrixA[i + N/2][j]
              A22[i][j] <- MatrixA[i + N/2][j + N/2]

              B11[i][j] <- MatrixB[i][j]
              B12[i][j] <- MatrixB[i][j + N/2]
              B21[i][j] <- MatrixB[i + N/2][j]
              B22[i][j] <- MatrixB[i + N/2][j + N/2]
```

Strassen Algorithm

```
// CONQUER: here we calculate P1...P7 matrices
    P1 <- STRASSEN(A11, B12-B22) //P1=A11(B12-B22)
    P2 <- STRASSEN(A11+A12, B22) //P2=(A11+A12)B22
    P3 <- STRASSEN(A21+A22, B11) //P3=(A21+A22)B11
    P4 <- STRASSEN(A22, B21-B11) //P4=A22(B21-B11)
    P5 <- STRASSEN(A11+A22, B11+B22) //P5=(A11+A22)(B11+B22)
    P6 <- STRASSEN(A12-A22, B21+B22) //P6=(A12-A22)(B21+B22)
    P7 <- STRASSEN(A11-A21, B11+B12) //P7=(A11-A21)(B11+B12)

      // calculate the result submatrices
    C11 <- P5 + P4 - P2 + P6
    C12 <- P1 + P2
    C21 <- P3 + P4
    C22 <- P5 + P1 - P3 - P7

      // MERGE: put them together and make our resulting Matrix
    for i <- 0 to N/2
       for j <- 0 to N/2
         MatrixResult[i][j] <- C11[i][j]
         MatrixResult[i][j + N/2] <- C12[i][j]
         MatrixResult[i + N/2][j] <- C21[i][j]
         MatrixResult[i + N/2][j + N/2] <- C22[i][j]
    return MatrixResult
```

Strassen Algorithm

Use the recursion function to evaluate the time complexity of the algorithm. For $n \geqslant 2$,

$$
\begin{aligned}
T(2n) &= 7\,T(n) + \Theta(n^2) \\
\log_2 n = k \Rightarrow\ T(k+1) &= 7\,T(k) + \Theta(2^{2k}) \\
\Rightarrow\ T(n) &= O(n^{\log_2 7}) \\
&\approx O(n^{2.81}) < \Theta(n^3)
\end{aligned}
\tag{2}
$$

Intel Math Kernel Library

Intel Math Kernel Library

The **MKL** is a optimized implementation of many math functions exclusively on x86 architecture and processors supports the Intel SIMD instructions, especially in Intel(R) processors. This library is implemented by assembly codes and C++ codes that are extremely optimized by Intel SIMD instructions(**SSE**,**AVX**), in order to maximize performance on matrix operations.

Performance of Intel MKL

The Intel MKL computes the 2500*2500 matrix multiplication within 6000 milliseconds.

```
Rows of A:500, Cols of A:2432
Rows of B:2432, Cols of B:500 time=308.777800 ms
Rows of A:700, Cols of A:2789
Rows of B:2789, Cols of B:700 time=548.609200 ms
Rows of A:900, Cols of A:2168
Rows of B:2168, Cols of B:900 time=553.480600 ms
Rows of A:1100, Cols of A:2793
Rows of B:2793, Cols of B:1100 time=835.885200 ms
Rows of A:1300, Cols of A:2073
Rows of B:2073, Cols of B:1300 time=815.472500 ms
Rows of A:1500, Cols of A:2293
Rows of B:2293, Cols of B:1500 time=1045.739900 ms
Rows of A:1700, Cols of A:2738
Rows of B:2738, Cols of B:1700 time=1399.515300 ms
Rows of A:1900, Cols of A:2189
Rows of B:2189, Cols of B:1900 time=1349.106500 ms
Rows of A:2100, Cols of A:2462
Rows of B:2462, Cols of B:2100 time=1649.516100 ms
Rows of A:2300, Cols of A:2453
Rows of B:2453, Cols of B:2300 time=1809.381100 ms
Rows of A:2500, Cols of A:2916
Rows of B:2916, Cols of B:2500 time=2278.926100 ms
```

Contents

Blocks

Lorem Ipsum

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Observation

Simmons Dormitory is composed of brick.

Contents

OCaml Code

Paragraph function

Write a function 'paragraph' that constructs a picture of width w of some text t, such that the content splits into as many lines as needed to fit into a paragraph of w columns.

paragraph.ml

```ocaml

let paragraph s n =
  let rec traverse buffer n i = function
   | [] -> Picture.row (Lst.reverse buffer)
   | x::xs ->
   if i = n then traverse (x::'\n'::buffer) n 1 xs
   else traverse (x::buffer) n (i+1) xs in
   traverse [] n 0 (Strng.of_string s);;
```

Thank you for your attention!

Contents

Backup-Slide