**矩阵乘法Cache/分片/SSE/AVX**

**Cache**

|  |
| --- |
| C++ void trans\_mul(int n, float a[][maxN], float b[][maxN], float c[][maxN]) {  // Cache优化（矩阵转置）  for (int i = 0; i < n; ++i)  for (int j = 0; j < i; ++j)  swap(b[i][j], b[j][i]);   for (int i = 0; i < n; ++i) {  for (int j = 0; j < n; ++j) {  c[i][j] = 0.0;  for (int k = 0; k < n; ++k) {  c[i][j] += a[i][k] \* b[j][k];  }  }  }   // Transpose back matrix b  for (int i = 0; i < n; ++i)  for (int j = 0; j < i; ++j)  swap(b[i][j], b[j][i]); } |

**SSE**

|  |
| --- |
| C++ void sse\_mul(int n, float a[][maxN], float b[][maxN], float c[][maxN]) {  \_\_m128 t1, t2, sum; //float数据类型   // Transpose matrix b  for (int i = 0; i < n; ++i)  for (int j = 0; j < i; ++j)  swap(b[i][j], b[j][i]);   for (int i = 0; i < n; ++i) {  for (int j = 0; j < n; ++j) {  c[i][j] = 0.0;  sum = \_mm\_setzero\_ps(); //创建一个所有元素为0的128位XMM寄存器   // Sum every 4 elements  for (int k = n - 4; k >= 0; k -= 4) {  t1 = \_mm\_loadu\_ps(a[i] + k); //将打包的单精度浮点数加载到寄存器中（无需对齐）  t2 = \_mm\_loadu\_ps(b[j] + k);   t1 = \_mm\_mul\_ps(t1, t2); //将打包的单精度浮点数相乘  sum = \_mm\_add\_ps(sum, t1); //将两组打包的（通常是4个）单精度浮点数相加，每组中的元素分别相加，结果存在一个XMM寄存器中  }   sum = \_mm\_hadd\_ps(sum, sum); //将打包的单精度浮点数水平相加  sum = \_mm\_hadd\_ps(sum, sum);  \_mm\_store\_ss(c[i] + j, sum); //将两个单精度浮点数存储在寄存器中   // Handle the last n%4 elements  for (int k = (n % 4) - 1; k >= 0; --k) {  c[i][j] += a[i][k] \* b[j][k];  }  }  }   // Transpose back matrix b  for (int i = 0; i < n; ++i)  for (int j = 0; j < i; ++j)  swap(b[i][j], b[j][i]); } |

**SSE分块**

|  |
| --- |
| C++ void sse\_tile(int n, float a[][maxN], float b[][maxN], float c[][maxN]) {  \_\_m128 t1, t2, sum;  float t;   // Transpose matrix b  for (int i = 0; i < n; ++i)  for (int j = 0; j < i; ++j)  swap(b[i][j], b[j][i]);   for (int r = 0; r < n / T; ++r)  for (int q = 0; q < n / T; ++q) {  for (int i = 0; i < T; ++i)  for (int j = 0; j < T; ++j)  c[r \* T + i][q \* T + j] = 0.0;   for (int p = 0; p < n / T; ++p) {  for (int i = 0; i < T; ++i)  for (int j = 0; j < T; ++j) {  sum = \_mm\_setzero\_ps();   // Sum every 4th elements  for (int k = 0; k < T; k += 4) {  t1 = \_mm\_loadu\_ps(a[r \* T + i] + p \* T + k);  t2 = \_mm\_loadu\_ps(b[q \* T + j] + p \* T + k);  t1 = \_mm\_mul\_ps(t1, t2);  sum = \_mm\_add\_ps(sum, t1);  }   sum = \_mm\_hadd\_ps(sum, sum);  sum = \_mm\_hadd\_ps(sum, sum);  \_mm\_store\_ss(&t, sum);  c[r \* T + i][q \* T + j] += t;  }  }  }   // Transpose back matrix b  for (int i = 0; i < n; ++i)  for (int j = 0; j < i; ++j)  swap(b[i][j], b[j][i]); } |