O documento a seguir irá apresentar o relatório de um trabalho da disciplina de Conceitos de Linguagens de Programação da UFPEL, o trabalho tem como objetivo otimizar o tempo de execução de um código, o mesmo visa realizar a multiplicação de uma matriz quadrada NxN, preocupando-se com acessos ao cache e utilizar instruções vetoriais.

Nesse Relatório será apresentado o código comentado, algumas opções utilizadas pela dupla para realizar as otimizações e será apresentado um gráfico de desempenho (Tempo), e um teste T de student para calcular o nível de confiança do resultado gerado.

O Código mostrado a seguir foi o resultante após as otimizações serem realizadas:

#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <immintrin.h>  
  
/\*comando para rodar com gprof  
 gcc -pg -o bla -O0 MultMatSemLeakAVX.c  
 ./bla 2048  
 gprof -b bla gmon.out  
\*/  
  
/\*comando para rodar avx  
 gcc -pg -mavx -o bla -O0 MultMatSemLeakAVX.c  
 ./bla 2048  
 gprof -b bla gmon.out  
\*/  
  
/\*comando para rodar valgrind  
 gcc -g -o bla -O0 MultMatSemLeakAVX.c  
 valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --num-callers=20 --track-fds=yes ./bla  
\*/  
  
int main(int argc, char \*argv[ ])  
{  
 int i,j,n,aux,resto,k;  
 n = atoi(argv[1]);  
 float \*\*mat1,\*\*mat2,\*\*mat3;  
  
 srand(time(NULL));  
  
 /\*

Como o malloc não tem o custo de zerar as posições, ele se torna mais rápido e como à duas primeiras matrizes são inicializadas com valores randômicos, não é necessário fazer atribuição de valores ao iniciá-las.  
 Já a terceira matriz que recebe o resultado é operada com a operação "+=" necessitando que o valor inicial seja zero, por isso ela é alocada com calloc.

\*/  
 mat1 = malloc(sizeof(float\*)\*n);  
 mat2 = malloc(sizeof(float\*)\*n);  
 mat3 = calloc(n , sizeof(float\*));  
 for (i = 0; i < n; i++)  
 {  
 mat1[i] = malloc(sizeof(float\*)\*n);  
 mat2[i] = malloc(sizeof(float\*)\*n);  
 mat3[i] = calloc(n, sizeof(float\*));  
 }  
  
 for (i = 0; i < n; i++)  
 {  
 for (j = 0; j < n; j++)  
 {  
 mat1[i][j] = rand()%RAND\_MAX;  
 mat2[i][j] = rand()%RAND\_MAX;  
 }  
 }  
  
 /\*Transpondo a segunda matriz podemos nos aproveitar da localidade espacial, assim podendo otimizar o uso da cache\*/  
 for (i = 0; i < n; i++)  
 {  
 for (j = i+1; j < n; j++)  
 {  
 if (j != i)  
 {  
 aux = mat2[i][j];  
 mat2[i][j] = mat2[j][i];  
 mat2[j][i] = aux;  
 }  
 }  
 }   
  
 \_\_m256 ymm0, ymm1;  
  
 for (i = 0; i < n; i++)  
 {  
 for (j = 0; j < n; j+=8)  
 {  
 if(j+8 <= n)  
 {

/\*Este if testa se ainda é possível pegar mais oito valores da matriz, ele vem primeiro por ser o teste que acontece mais vezes\*/  
 for (k = 0; k < n; k++)  
 {  
 ymm0 = \_mm256\_loadu\_ps(&mat1[i][j]);  
 ymm1 = \_mm256\_loadu\_ps(&mat2[k][j]);  
 ymm0 = \_mm256\_mul\_ps(ymm0, ymm1);  
 mat3[i][k] += ymm0[0] + ymm0[1] + ymm0[2] + ymm0[3]+ ymm0[4]+ ymm0[5]+ ymm0[6] + ymm0[7];  
 }  
 }  
 else  
 {

/\*Se não for possível pegar oito valores calcula-se sequencialmente\*/  
 resto = n%8;  
 for (aux=0; aux < resto; aux++)  
 {  
 for (k = 0; k < n; k++)  
 {  
 mat3[i][j+aux] += mat1[i][k] \* mat2[k][j+aux];  
 }  
 }  
 }  
 }  
 }  
  
 /\*Free em todas as matrizes para evitar vazamento de memória\*/  
 for (i = 0; i < n; i++)  
 {  
 free(mat1[i]);  
 }  
 free(mat1);  
 for (i = 0; i < n; i++)  
 {  
 free(mat2[i]);  
 }  
 free(mat2);  
 for (i = 0; i < n; i++)  
 {  
 free(mat3[i]);  
 }  
 free(mat3);  
  
 return 0;  
}

A dupla procurou alocar de maneiras mais corretas as matrizes a serem utilizadas, usando Malloc para as matrizes que serão inicializadas com um valor randômico, e calloc para a matriz que deve ser inicializado com 0, após gerar as matrizes, foi feita a transposta da segunda matriz para que possamos trabalhar com linhas e assim explorar a localidade espacial da cache no código, trazendo assim um ganho de desempenho no tempo de execução do mesmo, também foram utilizadas instruções SIMD para a vetorização da implementação, com isso foi possível executar oito operações simultaneamente, fazendo com que o ganho aumente ainda mais, por fim, foi realizado o free() das matrizes para que não houvesse vazamento de memória.

A seguir será apresentado dois gráficos com a média de tempo pelo número de entradas de cada matriz(NxN) dos dois códigos analisados, sem otimização e com otimização, foram obtidos 10 amostras para cada teste.



