Relatório Trabalho 1 - CAD

Aluno: Eric Reis Figueiredo

Máquinas usadas:

Pessoal: Intel Core i7
16Gb RAM DDR3
Ubuntu 17.04 (Gnome)

Saturno: especificações conhecidas.

Exercício 1)

a) Tempos de execução com compilador GNU foram medidos na minha máquina pessoal.

- **GNU**

Tamanho	-O0 (segundos)	-O1 (segundos)	-O2 (segundos)	-O3 (segundos)
10	5e-06	2e-06	2e-06	2e-06
100	0.003962	0.002304	0.000874	0.000849
500	0.590228	0.294501	0.135907	0.136098
1000	5.00214	2.39344	1.15735	1.22111

- **INTEL**

Tamanho	-O0 (segundos)	-O1 (segundos)	-O2 (segundos)	-O3 (segundos)
10				
100				
500				
1000				

b) A melhor ordenação foi a ordenação KIJ usando a minha máquina pessoal com o compilador GNU. Podemos tirar essa conclusão observando os seguintes resultados:

Flag: -O3 **N**: 1024

IJK: 1.22111 secs
JKI: 12.9543 secs
KIJ: 0.173171 secs
KJI: 12.837 secs

Exercício 2)

a) Se houve ganho de desempenho podemos considerá-lo desprezível. Nos dados coletados tivemos algumas vezes que apresentou melhora de até 0.18 segundos e outras vezes que apresentou piora de 0.1 segundos.

b) NB = 16. Porém apresentou desempenho um pouco pior do que a implementação normal. Com o programa compilado com a flag -O1, esta implementação apresentou uma piora de 1 segundo com NB = 2, mas a medida em que aumentei o NB para 4, 8 e 16 essa piora diminuiu. No caso de NB = 16 foi obtido o melhor desempenho desta implementação porém ainda pior do que o caso normal.

c) A minha implementação foi utilizar a técnica de NB apenas no laço *for* mais interno e nos laços mais externos desenrolar apenas 2 posições. Esta implementação apresentou desempenho melhor do que as duas implementações anteriores.

Com essa implementação obtive os seguintes resultados:

FLAG -O0
Tempo c/ implementação normal: 3.92618 segundos

NB	Combinação (segundos)
2	4.08222
4	2.63675
8	1.90622
16	1.52924

FLAG -O1

Tempo c/ implementação normal: 0.953887 segundos

NB	Combinação (segundos)
2	1.05181
4	0.65603
8	0.455659
16	0.342883

FLAG -O3

Tempo c/ implementação normal: 0.190123 segundos

NB	Combinação (segundos)
2	0.750653
4	0.465127
8	0.354996
16	0.207234

Devido a esses resultados é plausível supor que a otimização -O3 faça o desenrolar dos laços e algumas outras otimizações, já que obteve o melhor desempenho mesmo com a implementação normal.

d) Erro de compilação na máquina saturno (na minha máquina o erro não ocorre)

Exercício 3)

- **vec1.cpp**

    1. Dependência linha 15 e 16:
        a. Primeiramente imaginei que o compilador poderia achar que as posições dos vetores estivessem se confundindo na memória. Para tentar resolver esse problema fiz a alocação dinâmica dos vetores com o uso do *malloc()* porém isso não resolveu o problema.
        b. Como tinha certeza de que não havia dependência entre os vetores, utilizei *#pragma ivdep* para que o compilador ignorasse aquilo que ele estava considerando ser uma dependência. Isso resolveu a dependência porém não tornou o *loop* vetorizável pois a função *rand()* é chamada dentro do loop.
        c. Como o relatório de otimização diz que a função *rand()* não pode ser vetorizada acredito que o problema está nela. Poderia guardar o retorno dessa função em uma variável e utilizar a variável no interior do for, porém isso afetaria o resultado esperado e, por isso, optei por não realizar esta mudança.
    2. Non-unit Stride Loop:
        a. O relatório diz para explicitamente computar o contador de iteração antes de executar o loop. Para isso analisei o que era esperado do código. Reescrevi o *loop* utilizando dois *loops*, um para as posições menores que 10000, para as quais o contador é incrementado em 1 unidade e outra para as demais posições, para as quais o contador é incrementado em 50 unidades.

- **vec2.cpp**

    1. *setup(Type* table):*
        a. Nesse caso o loop não pode ser vetorizável pois a função *rand()* é chamada dentro dele.
    2. *loop for*:
        a. O *loop* externo não foi vetorizado pois o *loop* interno já estava sendo vetorizado e nenhum código é executado somente no *loop* externo. Adicionei *#pragma vector always*, mas mesmo assim a vetorização não é feita.

- **vec3.cpp**

    1. *setup(Type** table)*:
        a. O compilador estava considerando dependência na matriz. Após analisar o código, percebi que não ocorre tal dependência e para resolver adicionei *#pragma ivdep* nos dois *fors* mais internos.
    2. *main()*:
        a. O relatório reportou que a variável *table* estava com acesso desalinha no *loop for* mais interno. No entanto a variável *table* estava com acesso alinhado quando ela era utilizada dentro da função *setup()*. Isso fez com que eu percebesse que o que realmente estava desalinhado era a variável *mem*.
        b. Para resolver esse problema utilizei a instrução *_mm_malloc()* para alocar a memória de maneira alinhada e *_mm_free()* para liberar essa memória.
        c. Agora que eu tinha certeza que o acesso estaria alinhado utilizei *#pragma vector aligned* no *loop* mais interno.