Análise de Desempenho em Compilações

Instruções padrões e SIMD

*Introdução*

A compilação em algoritmos escritos em *C++* pode ser realizada de diversas formas, algumas delas com o intuito de se otimizar o código escrito, assim ganhando performance. Neste documento é feita uma comparação entre 3 tipos distintos de compilação de um código escrito em *C++*:

* Compilação sem otimizações
* Compilação com otimização da arquitetura AVX, com uso da flag

“*-mavx*”

* Compilação com otimização da arquitetura AVX2, com uso da flag

“*-mavx2*”

Assim, busca-se analisar as diferenças nos tempos de execução de cada versão, comparando os ganhos em performance das otimizações feitas com a compilação sem otimizações.

*O Código*

O código a ser compilado tem a função de informar tempos variados de execução de acordo com alguns parâmetros. No caso deste estudo, o tamanho de um vetor é variado até um limite “*max*”, e para cada tamanho é medido um tempo de execução em uma função. O tamanho do vetor aumenta com um passo “*plus*”. Assim, em um total de 6 funções diferentes, são calculados os tempos de execução para tamanhos de vetor variados.

O usuário deste algoritmo deve compilar o programa de 3 modos diferentes, para realização dos testes:

* “g++ func.cpp -o <nome> <size> <max> <plus>” : compilação sem otimização
* “g++ -O2 -ffast-math -ftree-vectorize -mavx func.cpp -o <nome> <size> <max> <plus>”: compilação com arquitetura *AVX*
* “g++ -O2 -ffast-math -ftree-vectorize -mavx2 func.cpp -o <nome> <size> <max> <plus>”: compilação com arquitetura *AVX* 2

Entende-se por <nome> o nome do arquivo compilado e <size> o tamanho inicial do vetor, que será usado nas funções.

Para os testes realizados neste relatório, os parâmetros foram: <size> foi inicializado com o valor de “*10000000*” (dez milhões), <max> com o valor de “*100000000*” (cem milhões), e <plus> com o valor de “*10000000*” (dez milhões). Portanto, para cada função serão medidos 10 tempos, para cada compilação. Não é recomendado um valor inicial em uma escala abaixo de 6 casas decimais, pois pode prejudicar a visualização dos resultados (empírico).

Para cada compilação, o programa irá gerar um arquivo em texto, chamado “*resultados.txt*”. Este arquivo informa o tempo de execução de cada função, para cada tamanho de vetor. Para melhor visualização destes resultados, o arquivo em questão pode ser importado no *Excel*, com o separador de números sendo ‘,’ (vírgula).

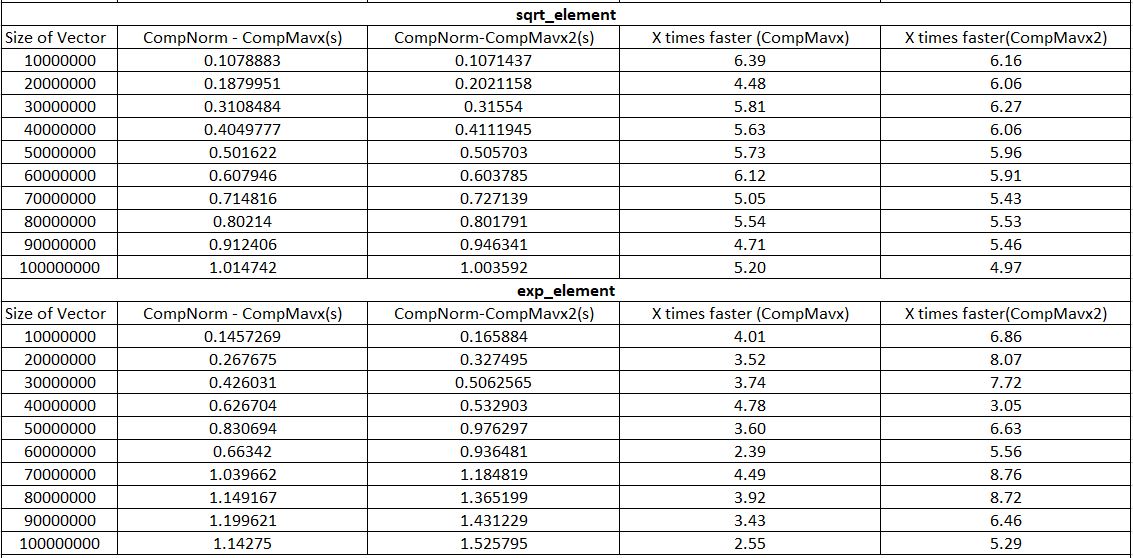
*Resultados*

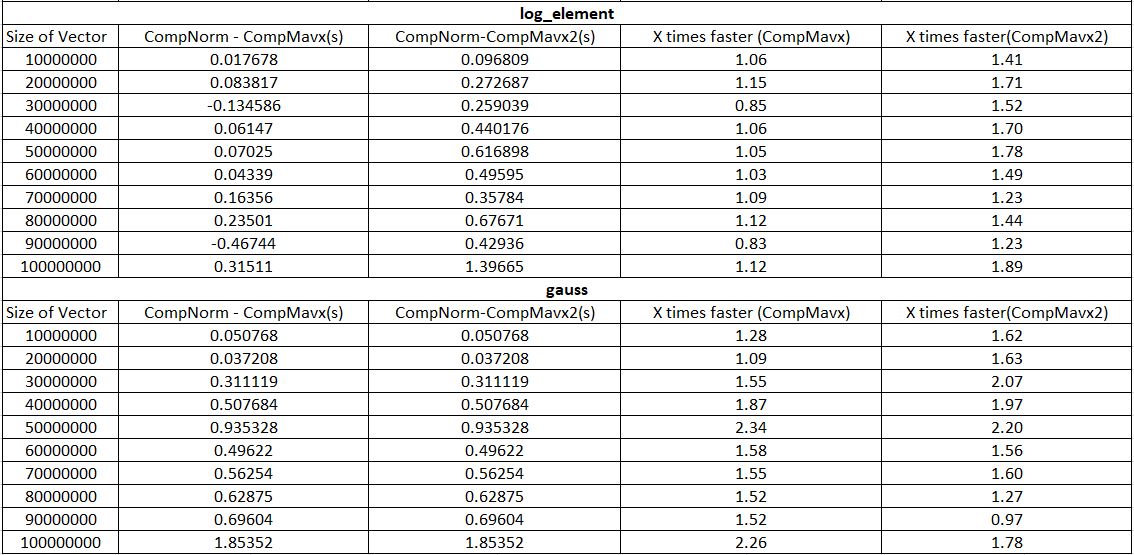
Dado os resultados pelo arquivo *“resultados.txt”,* pode-se plotar diversos gráficos comparativos entre os tempos obtidos. O intuito é comparar os tempos de execução da compilação sem otimização com os tempos otimizados pela arquitetura *AVX*. Para isso, o seguinte método foi utilizado:

1. Plotar o gráfico de cada função para cada compilação
2. Subtrair o tempo de compilação sem otimização dos tempos de compilação otimizados (*AVX e AVX2)*, para cada tamanho de vetor de cada função
3. Obter a porcentagem de otimização do tempo a partir das subtrações acima

Sabe-se que os dados que estão sendo trabalhados nos vetores são do tipo “double”, portanto é de se esperar que as compilações otimizadas consigam, aproximadamente, diminuir o tempo em 2 vezes, já que os registradores utilizados são do tipo “*%xmm#”* (128 bits), e inteiros possuem tamanho de 64 bits. Os resultados obtidos no primeiro teste foram:







Em ordem:

* 1ª coluna: Tamanho do Vetor
* 2ª coluna: Diferença de tempo entre compilação normal e compilação com arquitetura AVX
* 3ª coluna: Diferença de tempo entre compilação normal e compilação com arquitetura AVX2
* 4ª coluna: Quantidade de vezes que a compilação com arquitetura AVX foi mais rápida que a compilação sem otimização
* 5ª coluna: Quantidade de vezes que a compilação com arquitetura AVX2 foi mais rápida que a compilação sem otimização

*Discussão*

A melhora de desempenho depende do tamanho do vetor e do tipo de operação que está sendo efetuada (função).

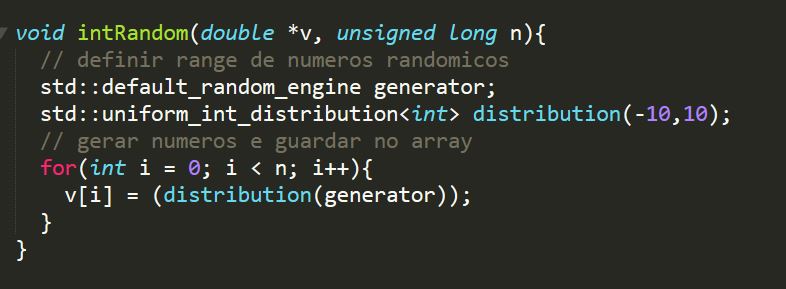
Nos testes, pode-se perceber um ganho de desempenho nas funções em geral, com algumas exceções onde o tempo de execução piorou com as otimizações dos compiladores (“*log\_element”*). A função “*sum\_positive”* e *“inner\_product”* mostraram ganhos em performance extremamente altos, enquanto o desempenho das funções “*log\_element*” *e “*gauss*”* dificilmente chegaram à melhora de velocidade esperada (2 vezes).

Assim, nota-se que o ganho de desempenho, em geral, bate com a teoria, onde se esperava uma melhora de aproximadamente 2 vezes ou mais em relação à compilação sem otimizações, dependendo da função. Além disso, observa-se uma melhora dos tempos medidos a partir do tamanho de vetores igual a *“10000000”* (10 milhões). As funções que melhor desempenharam com as otimizações, em geral, foram:

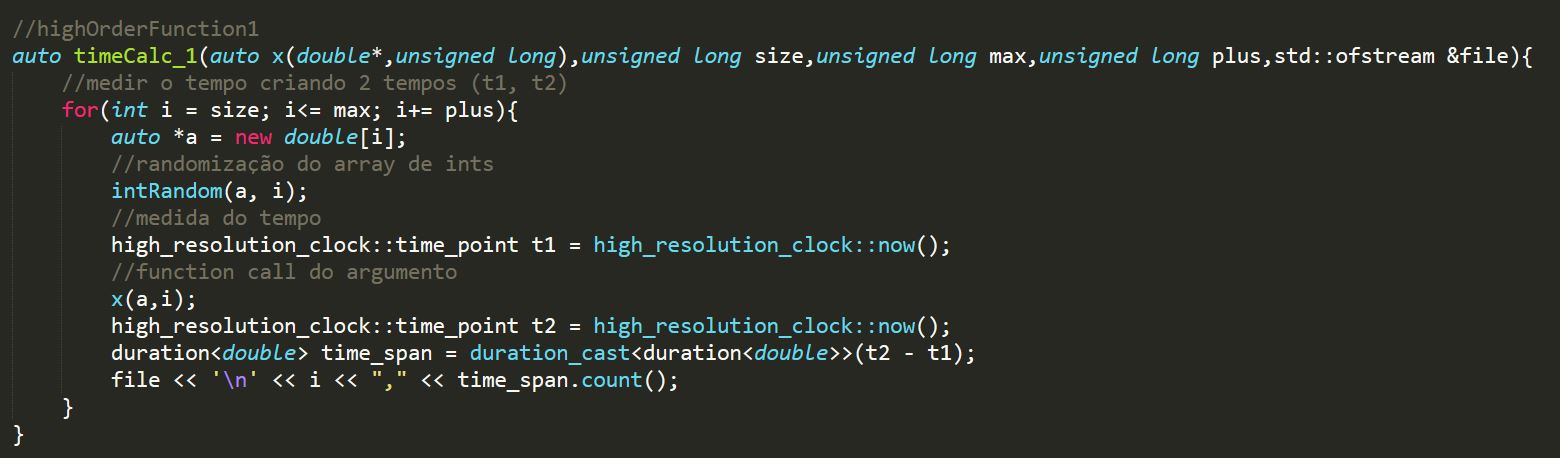
* ***“sum\_positive”***
* ***“inner\_product”***

Portanto, as operações de *soma* e *multiplicação* se mostraram mais rápidas.

*Imagens*

**

Função que gera vetores com inteiros randômicos



Uma das funções que realiza as "calls" de outras funções.