**Velocidade de execução do arquivo executável**

Já que estamos falando de técnicas de computação paralela, técnicas essas que visam reduzir o tempo de execução de uma determinada tarefa, não podemos nos esquecer de que, a par do paralelismo, há também a questão da velocidade de execução do arquivo executável que foi produzido pelo compilador.

O paralelismo reduz o tempo e a velocidade também o reduz. Então, se conseguirmos um bom grau de paralelismo e, também, produzir um executável que rode mais rápido, mais rápido ainda se completará a tarefa a ser executada.



Figura 1

A velocidade de execução de um arquivo executável depende de diversos fatores:

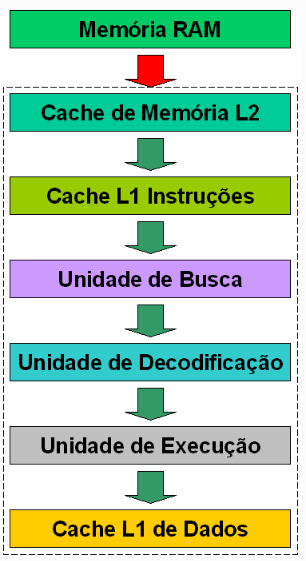
* **Linguagem de programação** - tirando o Assembly, as linguagens C/C++ são as que produzem os códigos mais rápidos.
* **Compilador** - para uma mesma linguagem, pode existir mais de um compilador e esses terem características diferentes, o que pode afetar na velocidade do executável. Também, para um dado compilador, pode haver opções de compilação que modificam a velocidade de execução, o que veremos adiante.
* **Hardware** - máquinas mais poderosas (CPU mais rápida, memória mais rápida, barramentos mais rápidos, mídias persistentes mais rápidas, etc.) executam, obviamente, mais rapidamente os programas.
  + Antes de se chegar aos limites a que se chegou atualmente quanto à velocidade das CPUs, a cada ano, a capacidade de processamento das máquinas dobrava ou mesmo quadruplicava. Quem possuía um Intel 386 (em torno de 1990) e adquiria um 486, percebia uma significativa mudança no desempenho dos seus aplicativos. Depois veio o Pentium, Pentium II, Pentium III, etc., as máquinas da linha AMD, e cada uma dessas gerações de máquinas estabelecia novos padrões nas expectativas dos usuários quanto aos tempos para abrir os aplicativos, executar as tarefas, etc.
  + Não só as CPUs são as responsáveis pela velocidade do processamento. As memórias também contam e muito. A memória, como qualquer dispositivo de armazenamento, tem um tempo de acesso ao seu conteúdo, ou seja, para a CPU ler um dado é preciso encadear um conjunto de sinais elétricos para que esse dado possa ser retornado. O tempo necessário para essa operação pode, por exemplo, impactar uma CPU rápida. As memórias mais comuns na atualidade são as DDR (Double Data Rate). As especificações principais são duas, a velocidade de clock que ela deve trabalhar e a taxa de transferência em MB/s. Por exemplo, DDR3‑1333/PC3200 - onde 1333 indica o clock de operação em MHz e 3200 a taxa de transferência em MB/s.
  + 

Figura 2 - extraída de https://www.clubedohardware.com.br/artigos/processadores/como-o-cache-de-mem%C3%B3ria-funciona-r34772/?nbcpage=4

* Para contornar as dificuldades de acesso à memória, criou-se a chamada memória cache, bem mais cara que a memória convencional, mas de acesso muito mais rápido. Essa memória armazena o que está para ser processado. Programas pequenos podem ser trazidos na íntegra para a cache, já os maiores são trazidos em partes.
* Por fim há que se considerar os barramentos. Canais de comunicação que fazem a interligação entre os diversos componentes do computador, transferindo dados de uns para os outros. Mais acessíveis aos especialistas em hardware, mas aos programadores é sempre bom saber que existem. Os barramentos também funcionam na base do clock, o que define a sua velocidade. Em essência um barramento (ou BUS) é uma via pública de dados onde tudo o que trafega deve ser encapsulado e conter, entre outros, a identificação do componente destinatário. "Se o pacote é meu, eu pego, se não é, ignoro". Barramentos mal projetados comprometem o desempenho de CPUs e memórias mais rápidas.

**Opções de compilação**

Como dito anteriormente, os compiladores geralmente oferecem opções que podem alterar as características do arquivo executável gerado.

No caso do g++ (também do gcc), existe a opção O ("ô" maiúsculo). Por meio desta opção podem-se gerar executáveis mais rápidos e/ou de menor tamanho. A rapidez (o que nos interessa aqui) está associada a como a CPU deve carregar o programa e posicionar as variáveis. Em linhas gerais, pode-se dizer que: ou a CPU decide como posicionar os componentes do programa, ou ela pode ser orientada a como posicioná-los. Obviamente que esta "orientação" pode ser atendida ou não de acordo com as disponibilidades dos registradores e das caches no momento. Uma segunda coisa é orientar a CPU a utilizar menos recursos lógicos/aritméticos, o que diminui a precisão nos cálculos, mas aumenta bastante a velocidade.

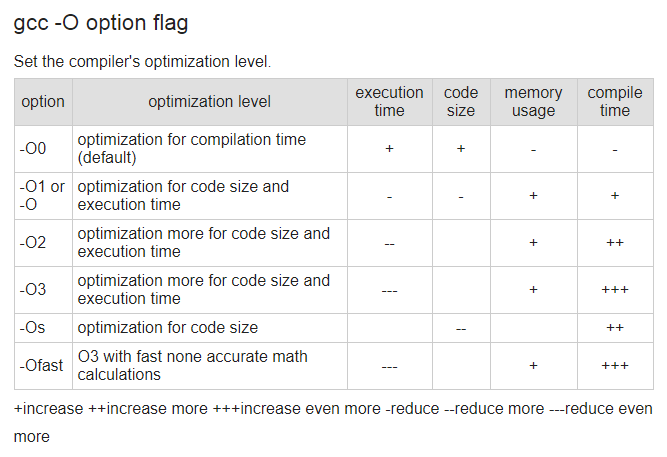


Figura 3 - extraída de https://www.rapidtables.com/code/linux/gcc/gcc-o.html

A sintaxe básica é a seguinte:

**g++ -Ox -o progexec main.cpp**