# Avaliação de Desempenho de Abordagens Paralelas Utilizando Curvas *B-spline*

**Vitor P. David**[1], **Douglas Custodio de Araujo**[1],
**Marcelo Zamith** [1], **Ubiratam de Paula** [1]

[1] Universidade Federal Rural do Rio de Janeiro (UFRRJ)

vitor.pito@gmail.com,custodiodouglas6@gmail.com

mzamith@ufrrj.br, upaula@ufrrj.br

***Resumo.*** *Tendo em vista a importância da paralelização de trechos do algoritmo, para a obtenção de desempenho, este artigo propõe a análise de desempenho para diferentes níveis de abordagens paralelas permitindo concluir a melhor abordagem para curvas b-spline. O desempenho foi avaliado utilizando diferentes níveis de paralelização, por meio de instruções, com a vetorização, e por meio de núcleos, através das threads. Nossos resultados demonstraram que uma abordagem através de núcleos obteve um aproveitamento muito bom enquanto que a vetorização não obteve ganhos.*

## 1. Introdução

Nos primeiros computadores, os códigos eram executados sequencialmente e com pouco ou quase nenhum recurso para acelerar o processamento. Nessa época, a única estratégia de aumentar a vazão de instruções executadas em um certo intervalo de tempo era reduzir o ciclo de *clock*, aumentando a frequência do processador. Foi uma estratégia que esbarrou em alguns limites físicos do processador.

Neste sentido, algumas técnicas foram adotadas visando aumentar a vazão de instruções. Dentre as técnicas, há a utilização de *threads* as quais permitem que trechos de um algoritmo sejam executados de forma paralela nos diversos núcleos do processador. Atualmente os processadores apresentam um número de threads que varia, onde há uma quantidade de threads físicas, atreladas aos núcleos físicos do processador, e threads virtuais que são emuladas em maior quantidade nos núcleos físicos.

Os registradores vetorizados também apresentam destaque, pois aumentam a vazão de instruções através do paralelismo de dados, ou seja, uma mesma instrução é executada sobre um conjunto de dados. Tendo surgido com uma tecnologia chamada MMX (*Multi Media Extensions*) e incluída nos processadores Pentium IV, que passaram a contar com registradores de 128bits e um conjunto de instruções vetorizadas. A aceleração alcançada chega dezesseis vezes o tamanho da palavra do registrador, considerando os registradores de 128bits que equipavam o Pentium IV e palavra de 8Bits. Atualmente, devido a avanços tecnológicos, tais registradores chegam a ter capacidade de 512Bits.

O outro lado do problema é a parte de software, neste sentido os compiladores surgem como um ferramenta fundamental, uma vez que realizam diversas otimizações que buscam de uma forma ou de outra melhorar o tempo de computação.

Além das otimizações, há disponível uma biblioteca que permite utilizar as instruções vetorizadas de forma direta, exigindo que os desenvolvedores modelem os dados para serem computados de forma vetorizada. A biblioteca é conhecida como *intrinsics*[1].

A vetorização realizada com base na biblioteca *intrinsics* apresenta um resultado melhor que a forma automática. Porque ao escrever o código utilizando tais instruções, o desenvolvedor precisa modelar seus dados na forma SIMD (Single Instruction Multiple Data). Já a biblioteca *pthread*[2] permite a utilização de threads, possibilitando a criação de diversas *threads* que serão distribuídas para os núcleos do processador para executarem paralelamente trechos do algoritmo.

Dentre os problemas que podem ser modelados em um formato SIMD, existe a construção de curvas b-spline. As curvas *b-spline* são amplamente utilizadas na computação gráfica para geração de curvas e superfícies suaves [Buss 2003]. As curvas b-spline são construídas a partir da aproximação de pontos, que são chamados como pontos de controle. E, o controle é local, onde cada seção da curva é ajustada por quatro pontos consecutivos.

Diversos trabalhos utilizam uma abordagem vetorizada para reduzir o tempo de computação e aumentar a vazão de instruções. Dentre esses trabalhos, alguns merecem destaque.

Em Stock et al. [Kevin Stock 2012], os autores discutiram uma solução para encontra a melhor forma de vetorizar um algoritmo, utilizando técnicas de inteligencia artificial e apresenta resultados onde a aceleração alcançada foi de 2x a 8x melhores que os compiladores da época.

Em Holewinski et al. [Justin Holewinski 2012] foi verificado principalmente a possibilidade de vetorização de uma aplicação através da criação de um grafo dinâmico de dependência de dados.

Bramas, B. [Bramas 2017] propôs a implementação vetorizada de um algoritmo clássico conhecido na literatura, Quick Sort. Nesse trabalho, foi utilizada instruções AVX-512 considerando os processadores Intel Skylake. Esse trabalho mostra que algoritmos antigos podem ser melhorados para se obter um ótimo desempenho em arquiteturas atuais.

Assim sendo, é proposto neste trabalho uma análise de performance do processador utilizando a construção de curva b-spline cúbica, considerando três abordagens de paralelismo: $i$) a primeira é utilizando biblioteca *intrinsics*. $ii$) a segunda abordagem é baseada em threads com a vetorização automática do compilador. E, $iii$) a última abordagem é baseada em threads com uso da biblioteca *intrinsics*. Na condução dos testes foram utilizados dois compiladores da linguagem C/C++. O GNU g++ disponibilizado de forma gratuita. E, o compilador proprietário da intel.

Este trabalho é dividido em seções, onde a Seção 2 descreve a abordagem paralela do problema, apresentando a estratégia para modelar o problema de forma a utilizar a biblioteca *intrinsics* e também a abordagem multithread. Os testes de performance são

[1] `https://software.intel.com/sites/landingpage/IntrinsicsGuide/#`

[2] `https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html`

apresentados e discutidos na Seção 3 e, por fim, as conclusões e trabalhos futuros são apresentados na Seção 4.

## 2. Abordagem paralela e vetorizada para geração de curvas b-spline cúbica

A curva b-spline faz parte de uma família de curvas conhecidas como splines [Buss 2003]. A principal característica dessa família em relação a outras é a construção da curva através da aproximação de pontos, que podem ou não serem igualmente espaçados e que são conhecidos como pontos de controle. Os pontos atuam de forma local, influenciando apenas uma seção da curva.

Assim, a b-spline cúbica pode ser definida por uma sequência de pontos de controle: $p_0$, $p_2$, $p_3$, ..., $p_n$, que junto com um conjunto de funções: $f_0(u)$, $f_1(u)$, $f_2(u)$, $f_3(u)$, ..., $f_n(u)$ definem um polinômio cúbico que descreve a curva $q(x) = \sum_{i=0}^{3} f_i(x) \times p_i$. Assim, o polinômio dado por $q(x)$ aproxima-se dos pontos $p_i$, mas não interpola como ocorre em outras curvas [Wolberg 1990, Buss 2003].

As funções $f_i(x)$ são definidas segundo suas propriedades [Wolberg 1990, Buss 2003], conforme o conjunto de Equações 1.

A construção da curva b-spline pode ser expressa matricialmente, onde as Equações 1 são colocadas na forma matricial (2).

$$
\begin{aligned}
f_0(x) &= \frac{1}{6}(1-x)^3 \\
f_1(x) &= \frac{1}{6}(3x^3 - 6x^2 + 4) \\
f_2(x) &= \frac{1}{6}(-3x^3 + 3x^2 + 3x + 1) \\
f_3(x) &= \frac{1}{6}(x^3)
\end{aligned}
\tag{1}
$$

$$
M = \frac{1}{6}\begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \tag{2}
$$

E, os pontos de controle são colocados na forma vetorial: $P = (x_i, x_{i+1}, x_{i+2}, x_{i+3})$ e a definição de um vetor $H = (t^3, t^2, t, 1)$. Desta forma, o algoritmo sequencial para construir a curva b-spline pode ser escrito como apresentado no Algoritmo 1.

**Algoritmo 1:** Algoritmo sequencial para obter a curva b-spline.

**Entrada:** Matriz $M$, vetor $H$, conjunto de pontos de controle $P$ e total de pontos $N$
**Saída:** Curva b-spline

**início**
  **para** $i + 3 < N$ **faça**
    **para** $(t = 0, t \leq 1, t = t + h)$ **faça**
      $\bar{x} \leftarrow \frac{(H \times M)}{6} \times \begin{bmatrix} X_i \\ X_{i+1} \\ X_{i+2} \\ X_{i+3} \end{bmatrix}$
      $\bar{y} \leftarrow \frac{(H \times M)}{6} \times \begin{bmatrix} Y_i \\ Y_{i+1} \\ Y_{i+2} \\ Y_{i+3} \end{bmatrix}$
    **fim**
    $i \leftarrow i + 1$
  **fim**
**fim**

A linha 2 do Algoritmo 1 define os quatro pontos de controle que são considerados para o cálculo do ponto $P(\bar{x}, \bar{y})$ da curva.

Para gerar uma curva longa, capaz de representar diversas formas, é necessário utilizar centenas de milhares de pontos de controle. Nesse caso, a geração da curva pode necessitar de um tempo maior de computação devido a grande quantidade de pontos de controle.

Assim, foram desenvolvidas três estratégias de paralelismo: o paralelismo a nível de instruções (biblioteca *intrinsics*); paralelismo baseado em threads, utilizando a biblioteca *pthread*; e, uma estratégia combinando a biblioteca *intrinsics* com a biblioteca *pthread*.

$i$) **- Estratégia baseada na biblioteca *intrinsics***: Esta abordagem baseou-se em realizar a multiplicação das matrizes de forma vetorizada, ou seja, as linhas e colunas são multiplicadas de quatro em quatro elementos.

$ii$) **- Estratégia baseada na biblioteca *pthread***: A abordagem baseada na biblioteca *pthread* é dividir o vetor $P$ entre as threads. Dessa maneira, cada thread executa $\frac{N-3}{T}$ vezes a operação $\frac{H*M*P}{6}$, sendo $T$ o número de threads. Para uma melhor comparação do problema, foi implementado a paralelização com duas, quatro e oito threads.

## 3. Testes realizados

Os testes realizados foram conduzidos num computador com um processador i7 com 4 núcleos físicos, *hyper thread*, 16Gb de memória principal e sistema operacional Ubuntu 18.04. O compilador GNU gcc utilizado foi na versão 8.3.0 e o compilador da intel foi na versão 19.0.3.199 20190206.

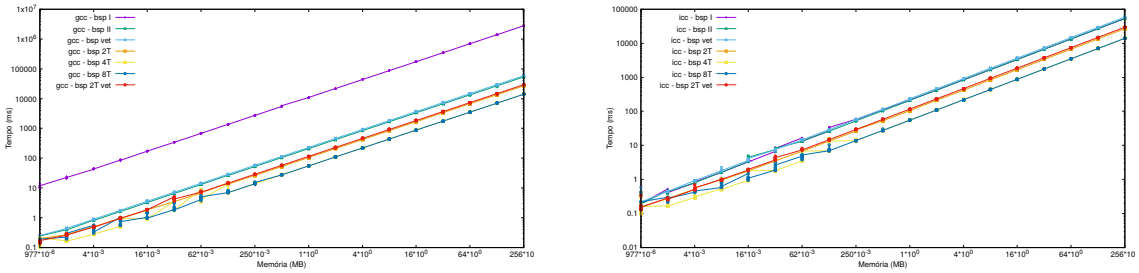
Para os testes, foi utilizado o parâmetro de otimização de código -O3 para ambos os compiladores. Para cada compilador, foram realizadas 10 execuções para as seguintes funções: **bsp I** que calcula a b-spline sem qualquer paralelização, mas utilizando a função **pow**, sendo a versão de referencia; **bsp II** realiza o mesmo cálculo substituindo a função **pow** por multiplicações; **bsp vet** é o cálculo da b-spline utilizando a vetorização; as versões multithread com duas, quatro e oito threads sem vetorização são: **bsp 2T**, **bsp 4T** e **bsp 8T**; por fim, a versão **bsp 2T vet** que combina as instruções vetorizadas da biblioteca *intrinsics* em duas threads.

As Figuras 2a e 2b ilustram os resultados alcançados com ambos os compiladores. A principal diferença significativa entre os resultados é na execução da função **bsp I**, onde o ICC obteve desempenho superior ao GCC. O ganho de tempo está na execução da função **pow** e este resultado é verificado comparando o tempo de execução das funções **bsp I** e **bsp II**.

Em relação as versões vetorizadas, o ganho obtido não foi significativo, sendo que o na versão do gcc houve um ganho um pouco maior por conta da ineficiência da execução da função **pow** na versão de referência **bsp I**.

A versão paralela alcançou o melhor resultado com oito threads em ambos os compiladores e com resultados muito próximos, sendo que o ganho não foi linear. Os resultados de oito e quatro threads foram bem parecidos, mas ainda sim nenhum versão apresentou ganho linear em relação a versão de referencia. Como já mencionado, o gcc

obteve uma aceleração maior em razão da ineficiência da função **pow**.




(a) GCC (b) ICC

**Figura 1. Resultados obtidos**

## 4. Conclusão e trabalhos futuros

É notável que abordagens paralelas para o processamento de algoritmos torna a execução otimizada. A abordagem de paralelização por *threads* obteve um ganho e os testes demonstraram que utilizar um número maior de *threads* que de núcleos é, entretanto, ineficiente, não gerando uma aceleração linear. Diante dos testes feitos, observou-se que a vetorização obteve um desempenho abaixo do esperado. Dessa forma, é necessário um melhor estudo sobre a vetorização através da análise da execução por meio de trabalhos futuros utilizando a biblioteca PAPI para haver a possibilidade de analisar possíveis inconsistências com a utilização de cache.

## Referências

Bramas, B. (2017). A novel hybrid quicksort algorithm vectorized using avx-512 on intel skylake. *International Journal of Advanced Computer Science and Applications(IJACSA)*, 8.

Buss, S. R. (2003). *3D computer graphics: a mathematical introduction with OpenGL*. Cambridge University Press.

Justin Holewinski, Ragavendar Ramamurthi, M. R. N. F. L.-N. P. A. R. P. S. (2012). Dynamic trace-based analysis of vectorization potential of applications. *ACM SIGPLAN Notices*, 47:371–382.

Kevin Stock, Louis-Noël Pouchet, P. S. (2012). Using machine learning to improve automatic vectorization. *ACM Transactions on Architecture and Code Optimization (TACO) - Special Issue on High-Performance Embedded Architectures and Compilers*, 8(50).

Wolberg, G. (1990). *Digital image warping*, volume 10662. IEEE computer society press Los Alamitos, CA.