CATEGORY: **DEVELOPER - TOOLS & LIBRARIES - DT02**

POSTER
P5184

CONTACT NAME
Nitin Agrawal: nitin.cic@gmail.com

GPU TECHNOLOGY CONFERENCE

GPU Accelerated B-spline coefficient Computation

Agrawal, Nitin[1,2], Gawali, Deepak[2,3], Nataraj S.V. , Paluri[2]

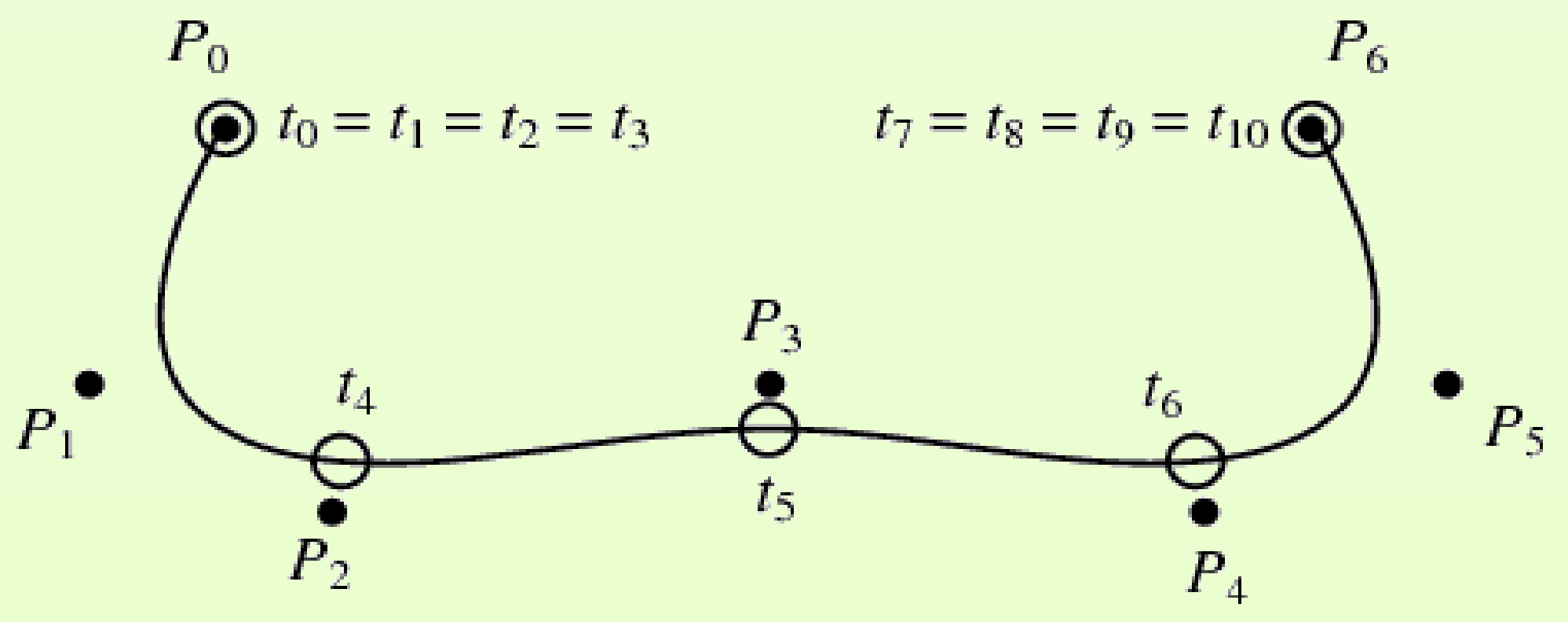
[1]Cluster Innovation Centre, University of Delhi, India
[2]Indian Institute of Technology, Bombay, India
[3]Vidyavardhini's College of Engineering and Technology, Maharashtra, India

Abstract

The work aims at investigating the use of Basic spline polynomial form in global polynomial optimization in an accelerated manner. The current work involves accelerated computation of B-spline coefficients corresponding to a polynomial in power form on a CPU environment. Furthermore the algorithm and methodology is accelerated on Graphics Processing unit for handling larger problems in a substantially less time. The parallelized GPU based approach offers substantial speed-ups over the CPU based implementation being run on a Dodeca-core processor.

Basic Splines

B-splines are the spline functions having minimal support for a given degree, smoothness and domain. These splines for a equidistant set of knots could be used for the purpose of curve fitting and otherwise. In the current scenarios B-splines are being used to estimate a polynomial function for achieving goals like computation of global optimization values amongst others`



A spline function of given degree for a given set of knots could be expressed as a linear combination of the B-splines of that degree. The thus computed B-spline is unique for a laid down set of knots.

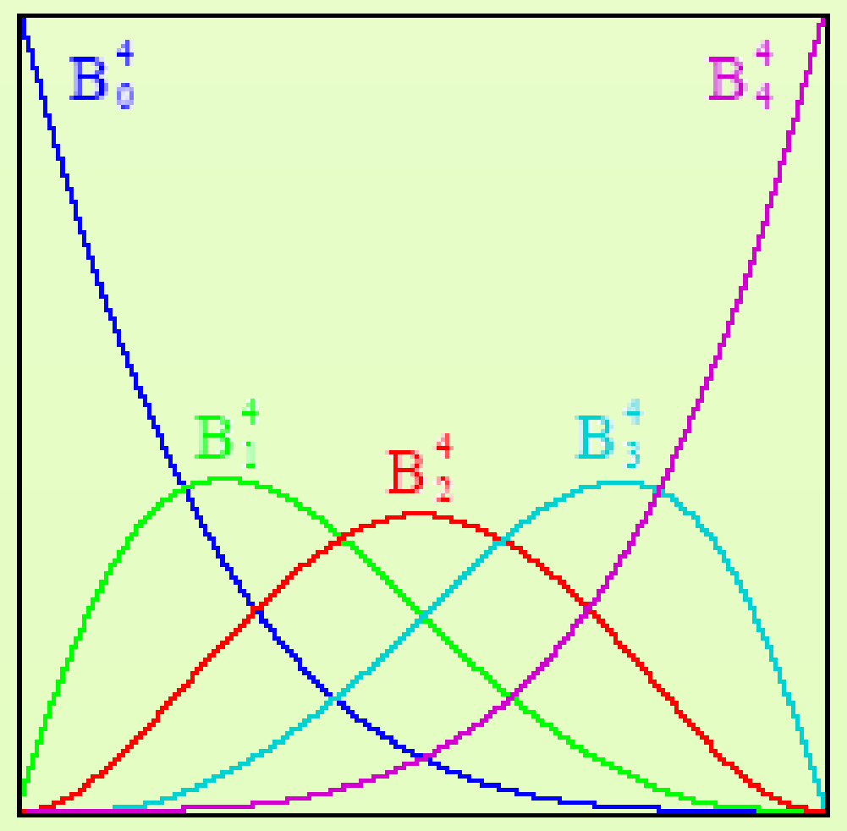
A spline can be formulated in the following manner

$$S_{k,t}(x) = \sum_i a_i B_{i,k}(x)$$

The polynomial can be further derived from the following recursive formulation

$$B_{i,1}(x) = \begin{Bmatrix} 1 \; if & t_i \leq x \leq t_{i+1} \\ 0 & otherwise \end{Bmatrix}$$

$$B_{i,k} = \frac{x-t_i}{t_{i+k-1}-t_i} B_{i,k-1}(x) + \frac{t_{i+k}-x}{t_{i+k}-t_{i+1}} B_{i+1,k-1}(x)$$



The process enables oneself to obtain the upper and lower bounds for a polynomial function. The largest value among the B-spline coefficients generated corresponding to a polynomial function can be considered to be an upper bound for that polynomial while the lowest value could be considered to be a lower bound . The repeated application of this process enables computation of the tight bounds.

Application of Basic Splines

- B-splines could be used to get a fair enough approximation of a multivariate polynomial of sufficiently high degree into splines of lower degrees which can ease the process of finding optimal points.
- Splines of lower degrees could reduce the sensitivity of the polynomial which can combat round-off errors to a great extent.
- B-splines are so formulated that each spline affects only limited number of control points which in turns allows a local control over the polynomial contrary to other curves fitting schemes such Bezier where the control in global and a modification in any fit affects all other points.

Images references: http://www.brnt.eu , http://www.math.uni-sb.de/

Algorithmic Implementation

```
Input:
  1. $coeffs$ : Polynomial Coefficient Matrix
  2. $degrees$ : A vector of degrees for each of the variables
  3. $segments$ : A vector of segments for each of the variables
  4. $domains$ : A vector of domains for each of the variables

Result: $bs\_coeffs$=B-spline Coefficient Matrix
initialization;
/* computing knot vectors and pi matrices                    */
for $i = 0$ to $no\_of\_vars - 1$ do
    knots=$find\_knot\_vectordegrees, segements, domains$ /* Parallel
        GPU Kernel Launch                                    */
    $find\_pi$ <<<
    $grid\_size, block\_size$ >>>($Pi\_matrices,knots,degrees,segment$)
end
$bs\_coeffs = coeffs$;
for $i = 0$ to $no\_of\_vars - 1$ do
    /* Multidimensional coefficient matrix is reshaped
        to a two dimensional matrix such that
        dimensional conformity w.r.t the pi matrix is
        maintained                                           */
    $Matrix\_multiplication(pi\_matrices[i], bs_coeffs)$;
    /* Parallel multiplication on GPU                        */
    $Matrix\_tranpose(bs\_coeffs)$;
    /* Parallel Transpose on GPU for n-dimensional
        transpose function                                   */
end
```

Algorithm 1: Parallel B-spline Coefficient Computation

```
Kernel $find\_pi$()
    Input:
      1. $knot\_vector$ : The knot vector
      2. $degree$ : Degree of the corresponding variable(s)
      3. $segments$ : No. of segments for the corresponding variable(s)

    Result: Pi matrix for the corresponding variable(s)
    initialization;
    /* Computation of the Pi Matrix using a thread for
        each pf the Pi element                               */
    $deg$ = threadIdx.x;
    $base$ = threadIdx.y;
    $knot\_part = knot\_vector(base + 1 : base + degree)$;
    /* compiling element(s) of the $knot\_vector$ to be used
        for computing the corresponding pi matrix
        element(s)                                           */
    if $deg == 1$ then
        $pi\_matrix[base][deg]$=1;
    else
        /* If the number of '1s' in the binary
            representation is equal to the degree then
            the $knot\_part$ elements corresponding to the
            indices of those '1s' are multiplied and
            added to the accumulator                         */
        for $i = 0$ to $2^{deg} - 1$ do
            $b_num = decimal2binary(i)$;
            if $ones(b\_num) == deg$ then
                indices=$fetch\_ones\_indices(b\_num)$
                $multi\_sum(knot\_part, indices)$
            end
        end
    end
```

Algorithm 2: Parallelized Algorithm for Pi Matrix generation

Experiments

System Configuration

CPU: Intel(R) Xeon(R) CPU E5-2620 @ 2.00HHz
GPU: NVIDIA Tesla K40

An example polynomial for which B-spline fit was computed

$$\begin{aligned} &100f^4 - 200f^2g + f^2 - 2f + 100g^4 - 200g^2h + 101\,g^2 - 2g + 100h^4 \\ &- 200h^2i + 101h^2 - 2h + 100i^4 - 200i^2j + 101i^2 - 2i + 100j^4 \\ &- 200j^2k + 100j^2 - 2j + 100k^4 - 200k^2l + 101k^2 - 2k + 100l^4 \\ &- 200l^2m + 101l^2 - 2l + 100m^4 - 200m^2n + 101m^2 - 2m + 100n^4 \\ &- 200n^2o + 101n^2 - 2n + 100o^2 + 9 \end{aligned}$$

No. Of Variables :10
Domain : $f, g, h, i, j, k, l, m, n, o \in [-5, 10]$
No. Of Segments : 2

Comparative Analysis

The table below portrays the comparison between the serial and GPU accelearated imlmentation for the above discusssed problem

	Matlab Code	C Code	CUDA C parallel code
Execution time (seconds)	Failed to respond	13.5	1.27

Conclusions

Accelaration of about 10.62 times was obtained on a GPU based implementation in comparison to CPU based implementation in C, while the Matlab version failed to produce meaningful results for the above discussed problem.

Future Works

Immediate future work involves conducting comparative experiments for benchmark problems in order to estimate the speedups on the GPU based implementation over C based serial CPU implementation in qualitative and quantitative terms . The algorithm so far designed furthermore needs to be incorporated into the optimization algorithm being worked upon for the purpose of global optimization

Moreover the GPU based implementation needs be compared against the CPU based parallel implementation on an OpenMP based platform.

References

- De Boor, C. (1972). On calculating with¡ b-splines. Journal of Approximation Theory, 6(1):50–62.
- Epperly, T. G. and Swaney, R. E. (1996). Branch and bound for global nlp: Iterative lp algorithm & results. In Global Optimization in Engineering Design, pages 37–73. Springer.
- Garloff, J. (1986). Convergent bounds for the range of multivariate polynomials. In Interval Mathematics 1985, pages 37–56. Springer.
- Garloff, J. and Smith, A. P. (2001). Solution of systems of polynomial equations by using Bernstein expansion. Springer.