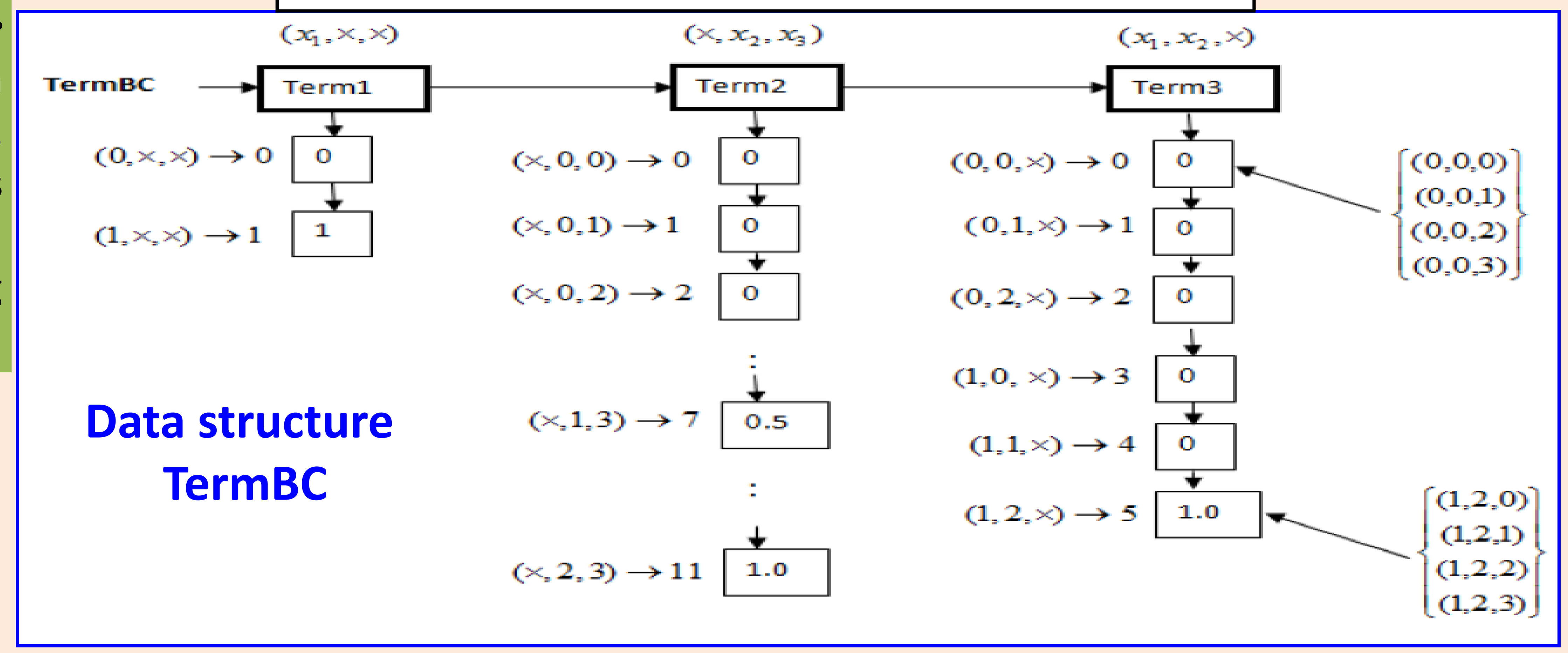
Modified Implicit Bernstein Form and its GPU Parallelization for computing the Bernstein Coefficients of a polynomial

P S Dhabe and P S V Nataraj,

Abstract:- We consider the problem of efficiently computing the Bernstein coefficients (BCs) of a polynomial on a box like domain. Recently, Smith [1] proposed the so-called Implicit Bernstein Form (IBF) for efficiently computing the BCs. The IBF is useful in polynomial global optimization solvers, see [2], [3]. In this work, we present a parallel version for computing the BCs, which we call as the Modified IBF, or simply, MIBF. In the MIBF, we can avoid many redundant computations, when each term contain only few variables, as typically is the case [4]. Using the MIBF, we obtained speedups of up to 84x for a 15 variable polynomial using NVIDIA's Tesla K20 and CUDA.

Motivation:- In the IBF mentioned above, if the domain spans several orthants, or if the variables occurring in polynomial terms fail to pass the *uniqueness*, *monotonicity*, or *dominance* tests, then we need to explicitly compute ***all*** the BCs, see [1]. That is, if each polynomial term contains only few variables, then IBF may involve many redundant computations. For instance, in the data structure *TermBC* described, we see that each BC of term3 is computed *4* times using IBF! We attempt to cut down on these redundant computations using the proposed MIBF. In MIBF, we use *TermBC* to compute BCs exactly as required. Computing all the BCs using MIBF can easily be parallelized on GPUs using NVIDIA's CUDA.

$$p(x) = x_1 + x_2 x_3^3 + x_1 x_2^2 \quad \text{on } [0,1]^3$$



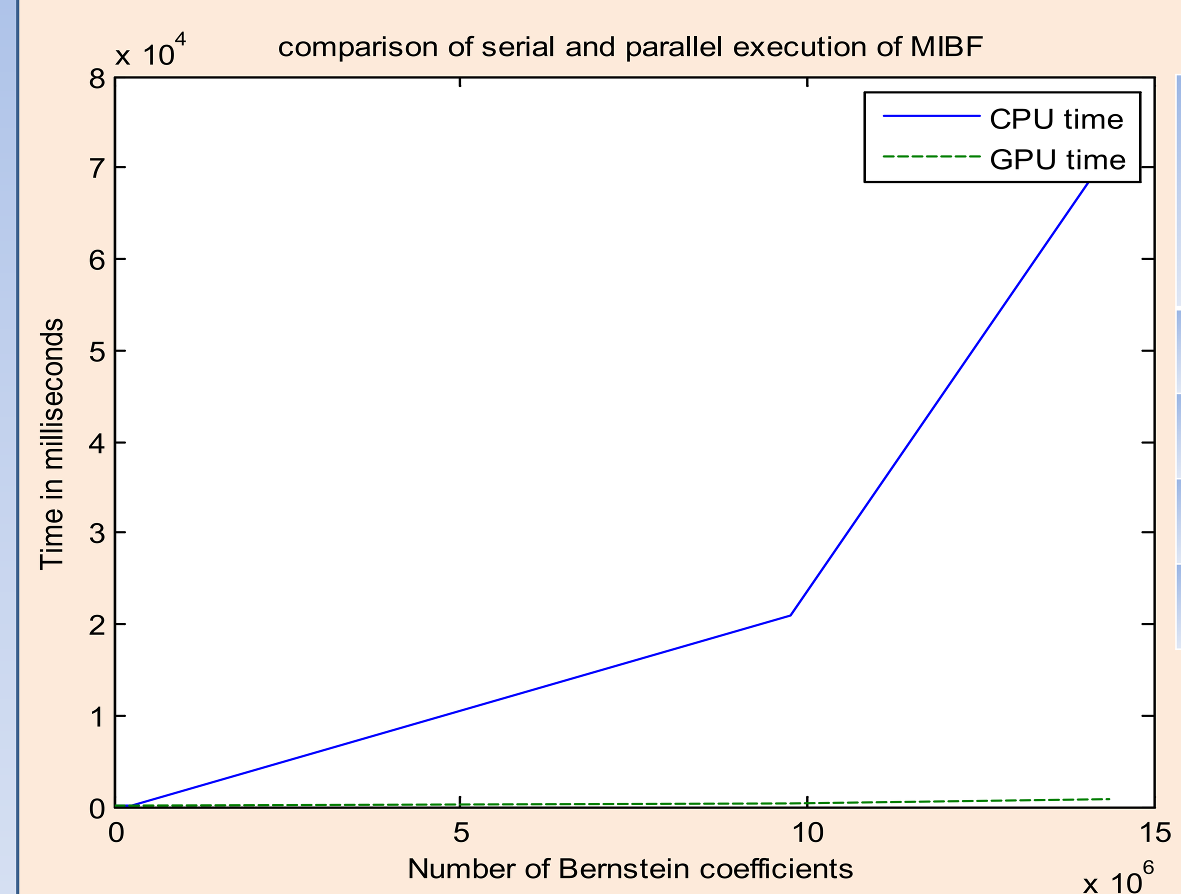
Data structure TermBC

Serial Algorithm of MIBF

Input :- $p(x)$ on domain x, variables l of maximum degree n, No. of terms t;
Output :- BC matrix B
begin
Step 1. Compute Implicit Bernstein Coefficients ($IBCs$) for $p(x)$ as in [1].
Step2. Compute $TermBCs$ and store it in linear memory.
Step3. Explicitly compute all the BCs
$for\ i = 1\ to\ (n+1)^l$
$sum = 0.0;$
$for\ \ j = 1\ to\ t$
Step3.1. Compute index q of i^{th} multiindex in $TermBC$ for j^{th} term
Step3.2. Add the q th value from $TermBC$ in sum
$sum = sum + TermBC[q];$
end
Step3.3. Store the computed BC at i^{th} location in CPU memory
$B[i] = sum;$
end
end

GPU parallelization of MIBF

We are parallelizing Step3 of serial algorithm, by launching $(n+1)^l$ CUDA threads. We store *TermBC* in GPU shared memory and computed BCs are stored in GPU global memory.



BCs	Serial time (A)	Parallel Time (B)	Speedup A/B
16807	1.9	**0.271**	**7.01**
262144	40.46	**1.57**	**25.77**
9765625	21049.95	**321.66**	**65.44**
14348907	71585.91	**850.05**	**84.21**

Comparison of serial and parallel Executions (msec.)

Platform Used

Intel's i7 3.06 GHz, Tesla k 20, Windows-7, VC++ on VS2010, CUDA toolkit- 5.5

Conclusion:-

The speedup obtained via GPU parallelization of MIBF is 84x , for a 15 variable polynomial. Serial time complexity of $O(n^{(l+1)})$ can be reduced to $O(tn)$ in CUDA parallel implementation.

References

[1] Smith A P, Fast construction of constant bound functions for sparse polynomials, Journal of Global Optimization, 45, 445-458, March 2009.
[2] J. Garloff, The Bernstein Algorithm, *Interval Computations*, 2:164-168, 1 993
[3] P. S. V. Nataraj and S. Ray, An efficient algorithm for range computation of polynomials using the Bernstein form, *Journal of Global Optimization*, 45:3, 403-426, Nov. 2009.
[4] J. Verschelde, The PHC pack, the database of polynomial systems, Technical report, University of Illinois, Mathematics department, Chicago, USA, 2001