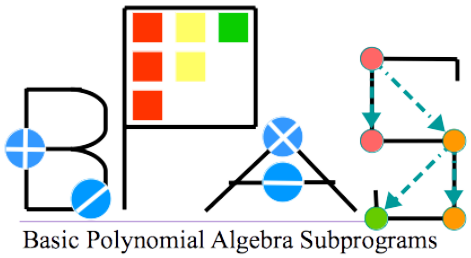
The Basic Polynomial Algebra Subprograms



Changbo Chen[1], Svyatoslav Covanov[2], Farnam Mansouri[3],
Robert H.C. Moir[4], Marc Moreno Maza[4], Ning Xie[4], Yuzhen Xie[5]
[1] CIGIT, Chinese Academy of Sciences, China; [2] LORIA, University of Lorraine, France;
[3] Microsoft, Vancouver, Canada; [4] ORCCA, University of Western Ontario, Canada;
[5] Critical Outcome Technologies, London, Canada

December 16, 2016

Abstract

The Basic Polynomial Algebra Subprograms (BPAS) provides arithmetic operations (multiplication, division, root isolation, etc.) for univariate and multivariate polynomials over common types of coefficients (prime fields, complex rational numbers, rational functions, etc.). The code is mainly written in `CilkPlus` [10] targeting multicore processors. The current distribution focuses on dense polynomials and the sparse case is work in progress. A strong emphasis is put on adaptive algorithms as the library aims at supporting a wide variety of situations in terms of problem sizes and available computing resources. The BPAS library is publicly available in source at `www.bpaslib.org`.

1 Design and Specification

Inspired by the Basic Linear Algebra Subprograms (BLAS), BPAS functionalities can be organized into three levels. At Level 1, one finds basic arithmetic operations that are specific to a polynomial representation or a coefficient ring, such as multi-dimensional FFTs/TFTs, univariate real root isolation. At Level 2, arithmetic operations are implemented for all types of coefficients rings supported by BPAS (prime fields, ring of integers, field of rational numbers and functions). Level 3 gathers advanced arithmetic operations, e.g. real root isolation of a zero-dimensional regular chain.

Level 1 functions are highly optimized in terms of locality and parallelism. In particular, the underlying algorithms are often nearly optimal in terms of cache complexity [6]. This is the case for our modular multi-dimensional FFTs/TFTs [13], modular dense polynomial arithmetic [14] and Taylor shift [4] algorithms.

At Level 2, the user can choose between algorithms minimizing work (at the possible expense of decreasing parallelism) and algorithms maximizing parallelism (at the possible expense of increasing work). For instance, five different integer polynomial multiplication algorithms are available: Schönhage-Strassen, 8-way Toom-Cook, 4-way Toom-Cook, divide-and-conquer plain multiplication and the two-convolution method. The first one has optimal work (i.e. algebraic complexity) but is purely serial due to the difficulties of parallelizing 1D FFTs on multicore processors. The next three algorithms are parallelized but their

parallelism is static, that is, independent of the input data size; these algorithms are practically efficient when both the input data size and the number of available cores are small, see [11] for details. The fifth algorithm [3] relies on modular 2D FFTs which are computed by means of the row-column scheme; this algorithm delivers a high scalability and can fully utilize fat computer nodes.

This variety of parallel solutions leads, at Level 3, to adaptive algorithms which select appropriate Level 2 functions depending on available resources (number of cores, input data size). An example is parallel real root isolation. Many procedures for this purpose are based on a subdivision scheme. However, on many examples, this scheme exposes limited opportunities for concurrent execution, see [4]. It is, therefore, essential to extract as much as parallelism from the underlying routines, such as Taylor shift computations.

One application of the BPAS library is integration of rational functions [17] by means of a symbolic-numeric method. The implementation uses: (1) the Lazard-Rioboo-Trager algorithm of [2] to integrate univariate rational functions over the rational numbers, and (2) numerical root isolation powered by the MPSolve library available at `http://numpi.dm.unipi.it/mpsolve`.

2 User Interface

Inspired by computer algebra systems like `AXIOM` [9] and `Magma` [1], BPAS makes use of type constructors so as to provide genericity, for instance `SparseUnivariatePolynomial (SUP)` can be instantiated over any BPAS ring. For efficiency consideration, certain polynomial type constructors, like `DistributedDense MultivariateModularPolynomial (DDMMP)`, are only available over finite fields in order to ensure that the data encoding of a `DDMMP` polynomial consists only of consecutive memory cells. For the same efficiency consideration, the most frequently used polynomial rings, like `DenseUnivariateIntegerPolynomial (DUZP)` and `DenseUnivariateRationalNumberPolynomial (DUQP)` are primitive types. In other words, `SUP<Integer>` and `DUZP` implement the same functionalities; however the implementation of the latter is further optimized. Figure 1 shows a subset of BPAS' tree of algebraic data structures. Class names started

```
#include <bpas.h>

using namespace std;
int main(int argc, char *argv[]) {
  /* Univariate Integer Polynomial Multiplication */
  DUZP a(128), b(128);
  a.read("a_input.dat"); b.read("b_input.dat");
  DUZP c = a * b;
  cout << "c = " << c << endl;

  /* Real Root Isolation */
  DUQP p; // p = x^{127} + 127
  p = (p + mpq_class(1) << 127)
    + mpq_class(127);
  Intervals boxes =
    p.realRootIsolate(1/20);

  /* Symbolic Numeric Integration */
  SparseUnivariatePolynomial<RationalNumber> f, g;
  f.one(); f.setVariableName("x");
  g = (polymomialParser("1+2*x+2*x^2"))^4;
  UnivariateRationalFunction<
    SparseUnivariatePolynomial<RationalNumber>, RationalNumber> h (f, g);
  h.realSymbolicNumericIntegrate(53);

  return 0;
}
```

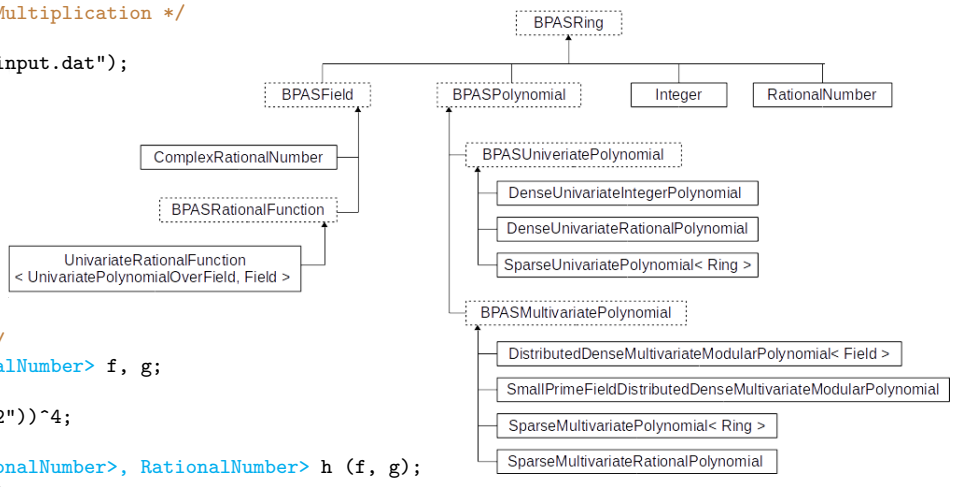


Figure 1: A snippet of BPAS code and a snapshot of BPAS data algebraic structures.

with 'BPAS' correspond to abstract classes, while others correspond to concrete classes. BPAS counts many other classes, for instance, `Intervals` and `RegularChains`.

3 Implementation techniques

Modular FFTs are at the core of asymptotically fast algorithms for dense polynomial arithmetic. Thus, a large body of code of the BPAS library is devoted to the computation of one-dimensional and multi-dimensional FFTs over finite fields. In the current release, the characteristic of those fields is of machine word size while larger characteristics are work in progress. The techniques used for the multi-dimensional FFTs are described in [13, 14] while those for the one-dimensional case is inspired by the FFTW project [5].

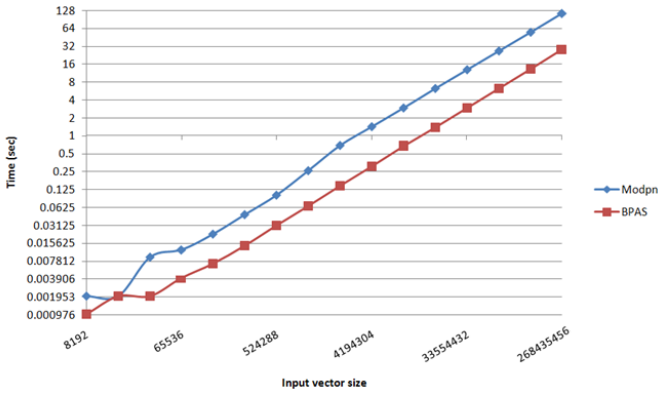


Figure 2: One-dimensional modular FFTs: Modpn vs BPAS.

BPAS one-dimensional FFTs code is optimized in terms of cache complexity and register usage. To achieve this, the FFT of a vector of size n is computed using a blocking strategy. That is, the corresponding FFT graph is "decomposed" (via tensor algebra) into blocks where each block is isomorphic to the FFT graph of an input vector of size K, where K is small, say $K = 16$. The value of K can be specified by the user or determined automatically when installing the library. At compilation time, this value is used to generate straight-line program (SLP) machine code. Instruction level parallelism (ILP) is carefully considered: vectorized instructions are explicitly used (SSE2, SSE4) and instruction pipeline usage is highly optimized. Various environment variables are available for the user to control different parameters in the code generation. Figure 2 compares running times (on Intel Xeon 5650) of one-dimensional modular FFTs computed by the Modpn library [16] and BPAS, both using serial C code in this case, where coefficients are in a prime field whose characteristic is a 57-bit prime.

Modular FFTs support the implementation of several algorithms performing dense polynomial arithmetic. An example is parallel multiplication of dense polynomials with integer coefficients by means of the *two-convolution method* [3].

4 Benchmarks

As mentioned above, one of the main purposes of the BPAS library is to take advantage of hardware accelerators and support the implementation of polynomial system solvers. With this goal, polynomial multiplication plays a central role. Moreover, both sparse and dense representations are important. Indeed, input polynomial systems are often sparse while many algebraic manipulations, like substitution, tend to densify data. Parallel sparse polynomial arithmetic has been studied by Gastineau and Laskar in [7] and by Monagan and Pearce in [12]. Up to our knowledge, BPAS is the first publicly available library for parallel dense polynomial arithmetic. For this reason, we compare BPAS' parallel dense polynomial multiplication against state-of-the-art counterpart implementation in FLINT 2.5.2 and Maple 2015.

Figure 3 shows the integer case. The input of each test case is a pair of polynomials of degree d where each coefficient has bit size N. Timings (in sec.) appear along the vertical axis. Two plots are provided:

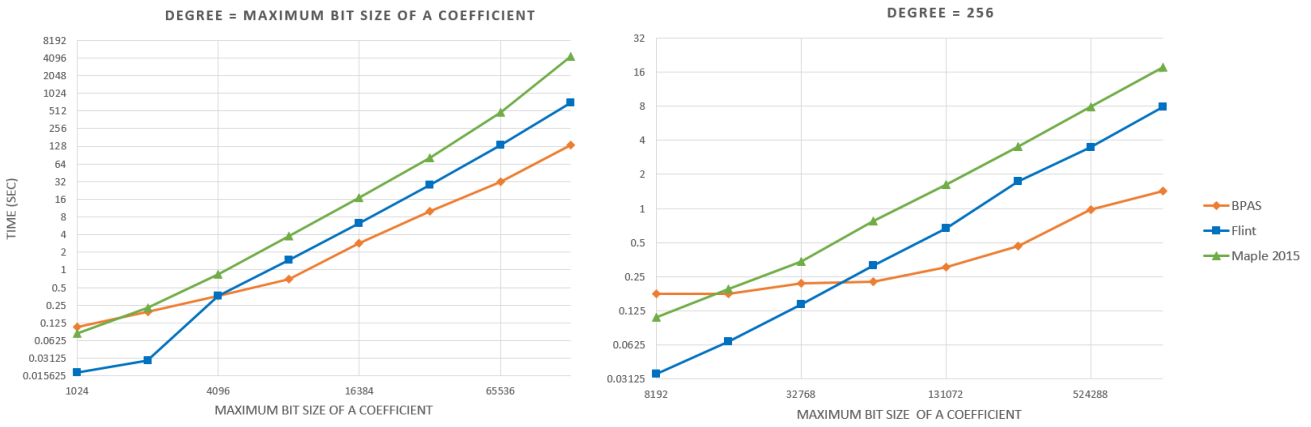


Figure 3: Dense integer polynomial multiplication: BPAS vs FLINT vs Maple.

one for which $d = N$ holds and one for d is much smaller than N. Our experimental results were obtained on an 48-core AMD Opteron 6168, running at 900Mhz with 256 GB of RAM and 512KB of L2 cache.

References

[1] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system I: The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997.

[2] M. Bronstein. *Symbolic Integration I – Transcendental Functions*. Springer-Verlag, 1997.

[3] C. Chen, S. Covanov, F. Mansouri, M. Moreno Maza, N. Xie, and Y. Xie. The Basic Polynomial Algebra Subprograms. In *Proc. ICMS*, LNCS, vol. 8592, p. 669-676, 2014.

[4] C. Chen, M. Moreno Maza, and Y. Xie. Cache complexity and multicore implementation for univariate real root isolation. *J. of Physics: Conference Series*, 341, 2011.

[5] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proc. of IEEE*, 93(2):216–231, 2005.

[6] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. *ACM Transactions on Algorithms*, 8(1):4, 2012.

[7] M. Gastineau and J. Laskar. Highly scalable multiplication for distributed sparse multivariate polynomials on many-core systems. In *CASC*, pages 100–115, 2013.

[8] W. Hart, F. Johansson, and S. Pancratz. FLINT: Fast Library for Number Theory `http://flintlib.org`.

[9] R. D. Jenks, R. S. Sutor. *AXIOM, The Scientific Computation System*. Springer-Verlag, 1992.

[10] C. E. Leiserson. The Cilk++ concurrency platform. *The Journal of Supercomputing*, 51(3):244–257, 2010.

[11] F. Mansouri. On the parallelization of integer polynomial multiplication. Master's thesis, The University of Western Ontario, London, ON, Canada, 2014. `www.csd.uwo.ca/~moreno/Publications/farnam-thesis.pdf`.

[12] M. B. Monagan and R. Pearce. Parallel sparse polynomial multiplication using heaps. In *Proceedings of ISSAC 2009*, pages 263–270. ACM, 2009.

[13] M. Moreno Maza and Y. Xie. FFT-based dense polynomial arithmetic on multi-cores. In *HPCS*, volume 5976 of *Lecture Notes in Computer Science*, pages 378–399. Springer, 2009.

[14] M. Moreno Maza and Y. Xie. Balanced dense polynomial multiplication on multi-cores. *Int. J. Found. Comput. Sci.*, 22(5):1035–1055, 2011.

[15] A. Schönhage and V. Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7(3-4):281–292, 1971.

[16] X. Li, M. Moreno Maza, R. Rasheed, and É. Schost. The modpn library: Bringing fast polynomial arithmetic into maple. *J. Symb. Comput.*, 46(7):841–858, 2011.

[17] R. H. C. Moir, R. M. Corless, D. J. Jeffrey. Unwinding paths on the Riemann sphere for continuous integrals of rational functions. In *Proc. of EACA*, 2014.