Fifteen years after DSC and WLSS2 What parallel computations I do today* [Invited Lecture at PASCO 2010]

Erich L. Kaltofen
Dept. of Mathematics, North Carolina State University
Raleigh, North Carolina 27695-8205,USA
kaltofen@math.ncsu.edu
http://www.kaltofen.us

ABSTRACT

A second wave of parallel and distributed computing research is rolling in. Today's multicore/multiprocessor computers facilitate everyone's parallel execution. In the mid 1990s, manufactures of expensive main-frame parallel computers faltered and computer science focused on the Internet and the computing grid. After a ten year hiatus, the Parallel Symbolic Computation Conference (PASCO) is awakening with new vigor.

I shall look back on the highlights of my own research on theoretical and practical aspects of parallel and distributed symbolic computation, and forward to what is to come by example of several current projects. An important technique in symbolic computation is the evaluation/interpolation paradigm, and multivariate sparse polynomial parallel interpolation constitutes a keystone operation, for which we present a new algorithm. Several embarrassingly parallel searches for special polynomials and exact sum-of-squares certificates have exposed issues in even today's multiprocessor architectures. Solutions are in both software and hardware. Finally, we propose the paradigm of interactive symbolic supercomputing, a symbolic computation environment analog of the STAR-P Matlab platform.

Categories and Subject Descriptors: I.1.2 [Symbolic and Algebraic Manipulation]: Algorithms; D.1.3 [Programming Techniques]: Parallel Programming

General Terms: algorithms, experimentation

Keywords: multiprocessor, multicore, memory bus contention, sparse interpolation, supersparse interpolation, parallel search, Lehmer's problem, single factor coefficient bound, interactive supercomputing

1. INTRODUCTION

1.1 A brief history of my research in parallel symbolic computation

In Fall 1982 as a newly minted Ph.D. hired by the theoretical computer science group at the University of Toronto, I was drawn into parallel computation. I shall briefly give an account of my research in parallel and distributed symbolic computation. The new parallel complexity class $\mathcal{NC}$ had been described by Stephen Cook (see [7]) and exact linear algebra problems were shown within it [5], in fact within uniform log-squared circuit depth. Since I worked on the problem of multivariate polynomial factorization, my first parallel result was on factoring polynomials over the complex numbers [24]. Because the integer GCD problem and the more general lattice basis reduction problem were (and are today) not known to be within $\mathcal{NC}$, the factorization of the defining polynomial for the field of definition of the complex factor coefficients had to be delayed until zero divisors appeared. Absolute irreducibility testing, however, was placed cleanly in $\mathcal{NC}$.[*] Independently, at the same time delayed factorization was used in sequential polynomial computation and called the D5 principle [12].

Work on polynomial matrix canonical forms with M. S. Krishnamoorthy and B. D. Saunders followed [29, 30], introducing random scalar matrix preconditioners for Smith form computation. Our general purpose algorithm for parallelizing straight-line programs for polynomials [47] and rational functions [25, Section 8] also increases the amount of total work of the parallel algebraic circuit, but our randomized algorithm for exact parallel linear system solving [37, 38], based on Wiedemann's algorithm [53], at last was processor-efficient, i.e., had only a poly-log factor more total work. No such solution is known today for Toeplitz systems or other structured systems (cf. [39]—a PASCO 1994 paper).

What was already apparent in the early 1990s, $\mathcal{NC}$ or even processor-efficient poly-logarithmic time algorithms are unrealizable on parallel computers: one does not have $O(n^3)$ processors for symbolic computation

[*] This material is based on work supported in part by the National Science Foundation under Grants CCF-0830347 and DMS-0532140.

[*] My paper [24] in its conclusion poses the problem of approximate polynomial factorization, which we only could solve 19 years later [17, 36].

PASCO 2010, 21–23 July 2010, Grenoble, France.
Copyright 2010 ACM 978-1-4503-0067-4/10/0007 ...$5.00.

tasks, and that without communication latency, for large n. Our 1991 implementation of a systolic polynomial GCD algorithm [6] on our own MasPar ("Massively Parallel") computer was slower then the sequential code on a top end workstation. The Thinking Machines Corporation filed for bankruptcy in 1994 and bit-wise parallelism disappeared. As far as I can judge, circuits of poly-logarithmic depth are only realized, on chip, for basic arithmetic operations such as look-ahead addition or Wallace-tree multiplication, which can be exploited for packed modular arithmetic with small moduli [15]. However, I still read today, in papers on exponential sequential algorithms in real algebraic geometry, for example, of "parallel polynomial-time" solutions. Those algorithms are not realizable.

The black box model for polynomials [40] and the parallel reconstruction algorithm via sparse interpolation seemed much more practicable—embarrassingly parallel. Thus in 1990 we began developing a run-time support tool, DSC (Distributed Symbolic Computation) [10, 8]. Several features in DSC seem unsupported in commonly used distributed run-time support tools, such as MPI/PVM. DSC could automatically distribute source code, produced possibly by the black box polynomial algorithms, to be *compiled remotely* before execution. There was a (weak) encryption protocol implemented to prevent malicious compilation requests on the remote computers. In today's world-wide hacker-attack environment our protocol no longer can meet required security standards. A second feature tried to model processor loads via time-series models and selectively choose computers with predicted short process queues [49]. We worked with the assumption that DSC was the *only daemon process* making such predictions on the LAN, as multiple such forecasters would possible select the same target computer. It appears to me that some of the instabilities in today's stock markets may be the result of similarly interfering prediction programs.

In the end, even my own Ph.D. students switched from our home made DSC to MPI, and our sparse multivariate polynomial interpolation algorithm in FoxBox is implemented with MPI [9, Section 2.8]. MPI was supported on a parallel computer available to us, the IBM SP-2. Our second application is the block Wiedemann algorithm [27] and Austin Lobo's implementation for entries in $\mathbb{Z}_2$ [45, 34, 35]. Lobo called his library WLSS (pronounced WiLiSyS—Wiedemann's Linear System Solver) and the SP-2 implementation WLSS2, on which systems arising from the RSA factoring challenged could be solved. Parallel symbolic computation had finally become reality. Those papers with Lobo should become, quite unpredictably, my last ones on the subject for 15 years, until this paper.

The second PASCO conference [21], for which I chaired the program committee. was held in Maui, Hawaii, before ISSAC 1997. Hoon Hong, the general chair, had broadened the subject to parallel automatic theorem proving. There were still several (strong) $\mathcal{NC}$ papers. Laurent Bernardin reported on a "massively" parallel implementation of Maple [4]. In contrast, Gaston Gonnet in his 2010 talk at Jo60, the conference in honor of Joachim von zur Gathen's 60th birthday, speculated that Maple's success was due in part that parallelism was *not* pursued in the early years.

On the other hand, Jean-Louis Roch, who attended our workshop "Parallel Algebraic Computation" at Cornell University in 1992 [55], with Thierry Gautier and others fully embraced parallelism and built several general purpose APIs and run-time systems: Athapascan-1 and Kaapi. The multicore multithreaded architectures of today make data parallel programming far beyond vector processing a reality. In 2007 Marc Moreno Maza single-handedly has revived the PASCO conference series. In his own symbolic computation research Moreno Maza systematically deploys parallel APIs, he also uses Cilk. A renaissance has begun. One paper at PASCO 2007 started at exactly the same place where we had left off: a parallel block Wiedemann algorithm [14].

1.2 Overview of results

We describe three separate topics. In Section 2 we investigate the important problem of sparse polynomial interpolation. Interpolation constitutes the reconstruction phase when computing by homomorphic images. The Ben-Or/Tiwari [3] breaks the sequentiality of Zippel's algorithm [54], but term exponent vectors need to be recovered from modular images. We give a method based on the discrete logarithm algorithm modulo primes p with smooth multiplicative orders $p-1$. Our algorithm can handle very large degrees, i.e., supersparse inputs [28], and thus allows for Kronecker substitution from many to a single variable.

In Section 3, we report on our year-long computer search for polynomials with large single factor coefficient bounds and small Mahler measure. Our search was successfully executed on multiprocessor MacPros. A second search, for sum-of-squares lower bound certificates in Rump's model problem [33, 43], for which each process requires several GBs of memory, was less successful on the older multiprocessor MacPros due to memory bus contention. Intel's new "Nehalem" architecture somewhat mitigates those issues.

In Section 4 we introduce the paradigm of interactive symbolic supercomputing.

2. SUPERSPARSE POLYNOMIAL INTERPOLATION

2.1 Term recovery by discrete logarithms

In [9, Section 4.1], as stated above, we implemented a modification of Zippel's [54] variable-by-variable multivariate sparse polynomial interpolation algorithm with modular coefficient arithmetic. The individual black box evaluation in each iteration are carried out in parallel, but the algorithm increases sequentially the number of variables in the interpolants. The algorithm works with relatively small moduli, and our subsequent hybridization [32] of the Zippel and Ben-Or/Tiwari [3] has further reduced the size of the required moduli.

An alternative that interpolates all variables in a single pass but requires a large modulus is already described in [26]. First, the multivariate interpolation problem for a black box polynomial $f(x_1, \ldots, x_n) \in$

$\mathbb{Q}[x_1, \ldots, x_n]$ is reduced to a univariate problem by Kronecker substitution $F(y) = f(y, y^{d_1+1}, y^{(d_1+1)(d_2+1)}, \ldots)$, where d_j is an upper bound on the degree of the black box polynomial in the variable x_j, which either is input or determined by a Monte Carlo method [40, Section 2, Step 1]. Each non-zero term $x_1^{e_{i,1}} \cdots x_n^{e_{i,n}}$ in f is mapped to y^{E_i} in F, where

$$E_i = \sum_j \Big(e_{i,j} \prod_{1 \le k \le j-1} (d_k + 1) \Big).$$

One interpolates $F(y)$ and recovers from the terms y^{E_i} and the bounds d_j all $e_{i,j}$.

The modulus p is then selected as a prime such that $p - 1 > \max_i E_i$ and that $p - 1$ is a smooth integer, i.e., has a factorization into small primes $q_1 \cdots q_l = p - 1$, where $q_1 = 2$. The prime p also must not divide any denominator in the rational coefficients of f, for which the black box call modulo p will throw an exception. For primes with smooth $p - 1$ there is a fast discrete logarithm algorithm [48] and primitive roots $g \in \mathbb{Z}_p$ are quickly constructed, for example by random sampling and testing. Such primes are quite numerous and easy to compute due to a conjectured effective version of Dirichlet's theorem: $\mu\, Q + 1$ is prime for $\mu = O((\log Q)^2)$ [20]. For instance,

$$37084 = \max_{m \le 12800} (\operatorname*{argmin}_{\mu \ge 1} (\mu 2^m + 1 \text{ is prime})).$$

The values $a_k = F(g^k)$, $k = 0, 1, 2, \ldots$ are linearly generated by the minimal polynomial

$$\Lambda(z) = \Big(\prod_{i=1}^{t} (z - g^{E_i}) \bmod p \Big),$$

where t is the number of non-zero terms [3]. First, the $a_k \bmod p$ are computed in parallel via the black box for f for the needed k, and then the generator is determined (see Section 2.2 below). With the early termination strategy the number of terms t and Λ can be computed from $2t{+}1$ sequence elements a_k by a Monte Carlo algorithm that uses a random g [32]. Second, the polynomial $\Lambda(z)$ is factored into linear factors over $\mathbb{Z}_p$. The factorization is a simple variant of Zassenhaus's method: The $\mathrm{GCD}(\Lambda(z + r), z^{(p-1)/2} - 1)$ for a random residue r splits off about half of the factors. The first division $(z^{(p-1)/2} - 1) \bmod \Lambda(z + r)$ can utilize the factorization of $p - 1$ or simply use repeated squaring, all modulo $\Lambda(z + r)$. The two factors of approximately degree $t/2$ are handled recursively and in parallel. Third, from the linear factors $z - g^{E_i} \bmod p$ the E_i are computed in parallel with Pohlig's and Hellman's discrete logarithm algorithm. Fourth, the term coefficients are computed from the a_k by solving a transposed Vandermonde system [31, Section 5], which essentially constitutes the fine-grain parallel operations of univariate polynomial (tree) multiplication and multipoint evaluation. Fifth, the rational coefficients are determined from the modular images by rational vector recovery [41, Section 4].

A drawback of the above algorithm is the need for a large prime modulus, and this was our reason for implementing Zippel's interpolation algorithm in FoxBox in 1996. Recently, discrete logs modulo word-sized primes could be utilized to recover the term exponents $e_{i,j}$ [23]. We briefly mentioned our idea in an unpublished manuscript on sparse rational function interpolation in 1988 [26], without realizing that the algorithm was actually polynomial in $\log(\deg f)$ under Heath-Brown's conjecture. Algorithms for such supersparse (lacunary) inputs are today a rich research subject, and an unconditional polynomial-time supersparse interpolation algorithm is given in [18]. Supersparse interpolation of shifted univariate polynomials is described in [19], and the difficult problem of supersparse rational function interpolation is solved for a fixed number of terms in [44]. All algorithms have highly speeded parallel variants, which hopefully will become commonly available in symbolic computation systems in the near future.

2.2 Scalar generators via block generators

In Section 2.1, a scalar linear generator was needed for a linearly generated sequence a_k. The classical solution is to deploy the Berlekamp/Massey algorithm [46]. As the new values a_k trickle in, the algorithm can in a concurrent thread iteratively update the current generator candidate. As a specialization of the extended Euclidean algorithm [13] or a specialization of the σ-basis algorithm [2, 32], the classical algorithm seems hard to parallelize, even in a fine-grain shared memory model.

An alternative is to employ the *matrix* Berlekamp/Massey algorithm [11, 42] for purpose of computing a *scalar* linear generator. We shall describe the idea by example. Suppose the degree of the scalar linear recursion is $t = 18$. The scalar Berlekamp/Massey algorithm determines the minimal linear generator $\Lambda(z)$ from the $2t$ sequence elements $a_0, \ldots, a_{35}$. Instead, we use a blocking factor of $b = 2$ and consider the sequence of 2×2 matrices

$$\begin{bmatrix} a_i & a_{9+i} \\ a_{9+i} & a_{18+i} \end{bmatrix}, \quad i = 0, 1, \ldots, 18. \tag{1}$$

Each entry in (1) is linearly generated by Λ, so the scalar generator of the matrix sequence is also Λ. Instead, we compute the minimal right matrix generator. The (infinite) block Hankel matrix

$$\begin{bmatrix} a_0 & a_9 & a_1 & a_{10} & \ldots \\ a_9 & a_{18} & a_{10} & a_{19} & \ldots \\ a_1 & a_{10} & a_2 & a_{11} & \ldots \\ a_{10} & a_{19} & a_{11} & a_{20} & \ldots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{2}$$

has rank $t = 18$, which is achieved at the $(t/b) \times (t/b) = 9 \times 9$ blocks (dimensions 18×18), because a row and column permutation transforms the block Hankel matrix into the scalar Hankel matrix which has exactly rank t (cf. [27, Proof of Proposition 3]). Therefore the 2×2 matrix generator polynomial $\Gamma(z)$ has degree 9, whose determinant is $\Lambda(z)$. The latter follows because the highest degree invariant factor of $\Gamma(z)$ is the scalar linear generator [42, Theorem 2] and the degree $\deg(\det \Gamma) = t$ because the determinantal degree equals the rank of (2).

The method uses $b - 1$ more scalar sequence elements, but $\Gamma(z)$ is found after $2\,t/b + 1$ matrix sequence elements. The block algorithm seems to have greater locality as the polynomials are of lower degree. It requires, however, the computation of $\det(\Gamma)$ via interpolation, and therefore may not be competitive with the classical scalar Berlekamp/algorithm for parallel supersparse interpolation, unlike the block Wiedemann linear system solver, where the computation of a_k depends on a_{k-1}. But one could think of a black box polynomial where $F(g^k)$ is derived with the help of $F(g^{k-1})$.

3. MULTIPLE PROCESSORS/CORES–SINGLE MEMORY BUS

In Summer and Fall 2007 we acquired 2 newly Intel-based Apple MacPros with multiple CPUs and multiple cores, on which we installed Ubuntu Linux. At that time we pursued two large computations: first the search for polynomials with large single factor coefficient bounds [33, Remark 4] and the related search for polynomials with small Mahler measure (Lehmer's problem), and second the search for exact SOS certificates in Rump's model problem [33, Section 3.2]. Both searches are embarrassingly parallel and we launched multiple command-line Maple 11 and later Maple 12 processes in Linux detachable "screen"s to achieve full processor utilization.

Our search throughout the year of 2008 for pseudo-cyclotomic polynomials, for which we also used an older Apple G5, yielded no new polynomials. For the record, our own largest single factor bound is given by the irreducible integer polynomial

$$F(z) = a_{37} z^{37} + \sum_{i=0}^{36} a_i (z^{74-i} + z^i) \quad \text{with}$$

$$\begin{gathered}
a_0 = 137244374035256035,\ a_1 = 6484943836830415168,\\
a_2 = 15319353170995914190 8,\\
a_3 = 241160750720080242490 7,\\
a_4 = 284529793856410798415 04,\\
a_5 = 268288753013473830301366,\\
a_6 = 2105372123573295644409420,\\
a_7 = 14138714883963898462151808,\\
a_8 = 82921677184320004630302040,\\
a_9 = 431329478501438585427465254,\\
a_{10} = 2014156747639672791329597498,\\
a_{11} = 8526069501131479222465282376,\\
a_{12} = 32979342592280651952625919221,\\
a_{13} = 117343525840400678593760923162,\\
a_{14} = 386220797646892832924725343578,\\
a_{15} = 1181540655003118732221772208453,\\
a_{16} = 3373539469466421210816098963801,\\
a_{17} = 9021882900472427122636284167235,\\
a_{18} = 22669166923589015905675502095077,\\
a_{19} = 53664552516356435243212903922539,\\
a_{20} = 119977087506448109730882947309906,\\
a_{21} = 253858560921214782055361032381920,\\
a_{22} = 509315086548136054993905167906615,\\
a_{23} = 970529828476535410874212141091985,\\
a_{24} = 1759154390161310454823643987118589,\\
a_{25} = 3036998343927337089144845248619604,\\
a_{26} = 4999639546259331050695892101743471,\\
a_{27} = 7856622081008872596932525992122154,\\
a_{28} = 11795932815522505668032581481982119,\\
a_{29} = 16934616571889545324916521185499766,\\
a_{30} = 23263087926382581409159452491840837,\\
a_{31} = 30596272432934117736562047551265003,\\
a_{32} = 38547804638104808779028751533424518,\\
a_{33} = 46541915513845439185592821100345340,\\
a_{34} = 53870405856198473740160765586055008,\\
a_{35} = 59790381075274084971155248471629182,\\
a_{36} = 63645538749721902787135528353527656,\\
a_{37} = 64984262804935950161468248039123157,
\end{gathered}$$

where $\|F(z)\|_\infty = \|F(-z)\|_\infty = a_{37}$ and $\|F(z) \cdot F(-z)\|_\infty = 18920209630100132696430504439191918$. Thus the single factor coefficient growth is

$$\frac{\|F(z)\|_\infty}{\|F(z)\, F(-z)\|_\infty} > 3.43464813.$$

The polynomial was constructed from the minimizers in Rump's model problem and we believed it to be the largest single factor bound known as David Boyd's construction only yielded ratios below 3. When presenting our bounds in Summer 2008, John Abbott showed us polynomials with much lower degree and coefficient size. His subsequent paper [1] contains an irreducible F with $\deg(F) = 20$, $\|F\|_\infty = 495$, and $\|F(z)\, F(-z)\|_\infty = 36$. Differences between nearby polynomials with large single (reducible) factor coefficient bounds yield pseudo-cyclic polynomials. Michael Mossinghoff's web site lists the top 100 non-cyclotomic irreducible polynomials with a small Mahler measure, the first being Lehmer's [http://www.cecm.sfu.ca/~mjm/Lehmer/]. In Figure 1 we list how many times we have discovered each of the top 50 polynomials, mostly those of high sparsity. Unfortunately, the search yielded no new polynomials, perhaps because we used relatively low degree minimizing polynomials.

In terms of parallel execution, in our Lehmer polynomials search we achieved a very good utilization of all available 16 cores. The same, surprisingly, was not true for our second search for sum-of-squares certificates for lower bounds in Rump's model problem [43, Section 3]. The main difference is that each SOS searche required a substantial amount of memory, about 5GB, due to the size of the arising matrices in the Newton optimizers. As Table 2 in [43] indicates, for $n = 17$ one command line Maple process required almost twice the time per iteration for a lesser lower bound, that because we executed it concurrently with second such independent command line Maple process since we had sufficient real memory for both. Reminiscent to parallel computing 15 years ago, the slowdown was caused by contention on the memory bus, which may be considered a hardware logic fault. Figure 2 depicts the 2007 MacPro, 4 cores with each L1 I cache: 32K, L1 D cache: 32K, L2 cache: 4096K, with one of the memory cards pulled out. In contrast, Figure 3 shows our new Intel "Nehalem" MacPro, 16 cores with each L1 I cache: 32K, L1 D cache: 32K, L2 cache: 256K, L3 cache: 8192K, and with the 32GB memory card and its massive dual controllers pulled out. Both cabinets have the same size. Note the large gray multipin interface at the bottom of the Nehalem card, which lies sideways on the cabinet. We could verify that for at least 2 processes the memory contention problem was solved.

Figure 1: Michael Mossinghoff's Top 50 pseudo-cyclotomic polynomials

MM's	deg	Mahler measure	count
1.	10	1.176280818260	2248
2.	18	1.188368147508	—
3.	14	1.200026523987	1
4.	18	1.201396186235	8804
5.	14	1.202616743689	105
6.	22	1.205019854225	10
7.	28	1.207950028412	—
8.	20	1.212824180989	4
9.	20	1.214995700776	—
10.	10	1.216391661138	198
11.	20	1.218396362520	1598
12.	24	1.218855150304	—
13.	24	1.219057507826	—
14.	18	1.219446875941	—
15.	18	1.219720859040	—
16.	34	1.220287441693	—
17.	38	1.223447381419	—
18.	26	1.223777454948	—
19.	16	1.224278907222	1779
20.	18	1.225503424104	35
21.	30	1.225619851977	—
22.	30	1.225810532354	—
23.	26	1.226092894512	17
24.	36	1.226493301473	—
25.	20	1.226993758166	194
26.	12	1.227785558695	77
27.	30	1.228140772740	—
28.	36	1.229482810173	—
29.	22	1.229566456617	1
30.	34	1.229999039697	—
31.	38	1.230263271363	—
32.	42	1.230295468643	—
33.	10	1.230391434407	27995
34.	46	1.230743009076	—
35.	18	1.231342769993	—
36.	48	1.232202952743	—
37.	20	1.232613548593	133
38.	28	1.232628775929	—
39.	38	1.233672001767	—
40.	52	1.234348374876	—
41.	24	1.234443834873	—
42.	26	1.234500336789	—
43.	16	1.235256705642	72
44.	46	1.235496042193	—
45.	22	1.235664580390	—
46.	42	1.235761099712	—
47.	32	1.236083368052	—
48.	32	1.236198469859	—
49.	32	1.236227922245	—
50.	40	1.236249557349	—

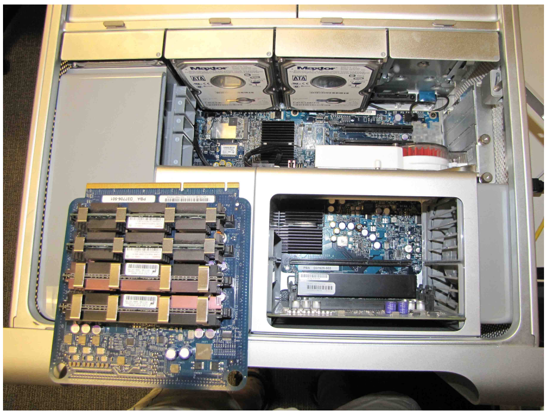

Figure 2: Dual processor dual core Xeon 3.0GH/7GB 2007 MacPro

Figure 3: Quad processor quad core Xeon 2.67GH/32GB 2009 Intel Nehalem MacPro

In addition, the authors of [16] report avoidance of bus contention by using Google's cached heap allocation scheme TCmalloc. Perhaps symbolic computation system vendors should also offer software compiled with such malloc schemes. We shall add that we also tried to tune the Maple garbage collection parameters.

Marc Moreno Maza has inquired with Maplesoft in response to our remarks, and a problem area seems to be the memory management strategy of Maple's garbage collector in a setting of parallel independent processes.

4. INTERACTIVE SYMBOLIC SUPERCOMPUTING

During my sabbatical at MIT in Spring 2006, with Alan Edelman we have investigated the use of genericity to create interfaces from symbolic computation platforms to Star-P servers [22] and other parallel implementations.

Laptops and desktops do not have hundreds of processors and large clusters of computers are housed in labs. The Internet makes it possible to access such high performance computers and networks from almost everywhere. The idea of interactive supercomputing is to place the data and computations of a Matlab or Mathematica/Maple/SAGE [52] session remotely on such a compute server and control the session from the local GUI interactively in such a way *that the supercomputing session is indistinguishable from what would be a locally run session.*

Alan Edelman's solution in Matlab is to overload the "`*`" operator and pseudo-postmultiply any value by a global variable `p` of a special type so that the resulting type is a reference to the remote storage. The relevant Matlab functions are then overloaded so that any arguments of the Star-P type delegate execution of the function to the remote supercomputer on the remote data.

In Figure 4 we give a code fragment how Maple's `LinearAlgebra` package could be overloaded for a special Star-P type. We anticipate to experiment with links to parallel implementations of the LinBox exact linear algebra library as soon as they are available. We add that the INTER*CTIVE supercomputing company was recently acquired by Microsoft. Note that SAGE's philosophy is to place its user interface above the symbolic computation system, while Star-P places the interface underneath it.

Acknowledgments: I thank Jean-Guillaume Dumas for sharing with me his experience with the memory bus contention problem and TCmalloc, and Marc Moreno Maza for his reviewing comments.

5. REFERENCES

[1] Abbott, J. Bounds on factors in Z[x]. *Mathematics Research Repository abs/0904.3057* (2009). URL: http://arxiv.org/abs/0904.3057.

[2] Beckermann, B., and Labahn, G. A uniform approach for fast computation of matrix-type Padé approximants. *SIAM J. Matrix Anal. Applic. 15*, 3 (July 1994), 804–823.

[3] Ben-Or, M., and Tiwari, P. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proc. Twentieth Annual ACM Symp. Theory Comput.* (New York, N.Y., 1988), ACM Press, pp. 301–309.

[4] Bernardin, L. Maple on a massively parallel, distributed memory machine. In Hitz and Kaltofen [21], pp. 217–222.

[5] Borodin, A., von zur Gathen, J., and Hopcroft, J. E. Fast parallel matrix and GCD computations. *Inf. Control 52* (1982), 241–256.

[6] Brent, R. P., and Kung, H. T. Systolic VLSI arrays for linear-time GCD computation. In *Proc. VLSI '83* (1983), pp. 145–154.

[7] Cook, S. A. A taxonomy of problems with fast parallel algorithms. *Inf. Control 64* (1985), 2–22.

[8] Díaz, A., Hitz, M., Kaltofen, E., Lobo, A., and Valente, T. Process scheduling in DSC and the large sparse linear systems challenge. *J. Symbolic Comput. 19*, 1–3 (1995), 269–282. URL: EKbib/95/DHKLV95.pdf.

[9] Díaz, A., and Kaltofen, E. FoxBox a system for manipulating symbolic objects in black box representation. In *Proc. 1998 Internat. Symp. Symbolic Algebraic Comput. (ISSAC'98)* (New York, N. Y., 1998), O. Gloor, Ed., ACM Press, pp. 30–37. URL: EKbib/98/DiKa98.pdf.

[10] Díaz, A., Kaltofen, E., Schmitz, K., and Valente, T. DSC A system for distributed symbolic computation. In *Proc. 1991 Internat. Symp. Symbolic Algebraic Comput. (ISSAC'91)* (New York, N. Y., 1991), S. M. Watt, Ed., ACM Press, pp. 323–332. URL: EKbib/91/DKSV91.ps.gz.

[11] Dickinson, B. W., Morf, M., and Kailath, T. A minimal realization algorithm for matrix sequences. *IEEE Trans. Automatic Control* AC-*19*, 1 (Feb. 1974), 31–38.

[12] Dicrescenzo, C., and Duval, D. *Le système D5 de calcul formel avec des nombres algébriques.* Univ. Grenoble, 1987, Doctoral Thesis by Dominique Duval, Chapter 1. Jean Della-Dora (Thesis Adisor).

[13] Dornstetter, J. L. On the equivalence between Berlekamp's and Euclid's algorithms. *IEEE Trans. Inf. Theory* IT-*33*, 3 (1987), 428–431.

[14] Dumas, J.-G., Elbaz-Vincent, P., Giorgi, P., and Urbanska, A. Parallel computation of the rank of large sparse matrices from algebraic K-theory. In *PASCO'07 Proc. 2007 Internat. Workshop on Parallel Symbolic Comput.* (2007), pp. 43–52.

[15] Dumas, J.-G., Fousse, L., and Salvy, B. Simultaneous modular reduction and Kronecker substitution for small finite fields. *J. Symbolic Comput. to appear* (2010).

[16] Dumas, J.-G., Gautier, T., and Roch, J.-L. Generic design of Chinese remaindering schemes. In *PASCO'10 Proc. 2010 Internat. Workshop on Parallel Symbolic Comput.* (New York, N. Y., 2010), M. Moreno Maza and J.-L. Roch, Eds., ACM.

[17] Gao, S., Kaltofen, E., May, J. P., Yang, Z., and Zhi, L. Approximate factorization of multivariate polynomials via differential equations. In *ISSAC 2004 Proc. 2004 Internat. Symp. Symbolic Algebraic Comput.* (New York, N. Y., 2004), J. Gutierrez, Ed., ACM Press, pp. 167–174. ACM SIGSAM's ISSAC 2004 Distinguished Student Author Award (May and Yang). URL: EKbib/04/GKMYZ04.pdf.

[18] Garg, S., and Schost, Éric. Interpolation of polynomials given by straight-line programs. *Theoretical Computer Science 410*, 27-29 (2009), 2659 – 2662.

[19] Giesbrecht, M., and Roche, D. S. Interpolation of shifted-lacunary polynomials. *Computing Research Repository abs/0810.5685* (2008). URL: http://arxiv.org/abs/0810.5685.

[20] Heath-Brown, D. R. Almost-primes in arithmetic progressions and short intervals. *Math. Proc. Camb. Phil. Soc. 83* (1978), 357–375.

Figure 4: Overloading a Maple package procedure

```
> LAstarp:=module()
>    export RandomMatrix;
>    local __RandomMatrix;
>    unprotect(LinearAlgebra:-RandomMatrix);
>    __RandomMatrix := eval(LinearAlgebra:-RandomMatrix);
>    LinearAlgebra:-RandomMatrix:=proc()
>       if type(args[1],string) then RETURN("Calling starp with " || args);
>                             else RETURN(__RandomMatrix(args));
>       fi;
>    end; # RandomMatrix
> end; # LAstarp
                     LAstarp := module() local __RandomMatrix; export RandomMatrix;  end module
> LinearAlgebra:-RandomMatrix(2,2,generator=0..1.0);
                                    [0.913375856139019393    0.905791937075619225]
                                    [                                            ]
                                    [0.126986816293506055    0.814723686393178936]
> LinearAlgebra:-RandomMatrix("overloading");
                                           "Calling starp with overloading"
```

[21] Hitz, M., and Kaltofen, E., Eds. *Proc. Second Internat. Symp. Parallel Symbolic Comput. PASCO '97* (New York, N. Y., 1997), ACM Press.

[22] INTER*CTIVE supercomputing. Star-P overview. Web page, 2008. URL http://www.interactivesupercomputing.com/.

[23] Javadi, S. M. M., and Monagan, M. On sparse polynomial interpolation over finite fields. Manuscript, 2010.

[24] Kaltofen, E. Fast parallel absolute irreducibility testing. *J. Symbolic Comput. 1*, 1 (1985), 57–67. Misprint corrections: *J. Symbolic Comput.* vol. 9, p. 320 (1989). URL: EKbib/85/Ka85_jsc.pdf.

[25] Kaltofen, E. Greatest common divisors of polynomials given by straight-line programs. *J. ACM 35*, 1 (1988), 231–264. URL: EKbib/88/Ka88_jacm.pdf.

[26] Kaltofen, E. Unpublished article fragment, 1988. URL http://www.math.ncsu.edu/~kaltofen/bibliography/88/Ka88_ratint.pdf.

[27] Kaltofen, E. Analysis of Coppersmith's block Wiedemann algorithm for the parallel solution of sparse linear systems. *Math. Comput. 64*, 210 (1995), 777–806. URL: EKbib/95/Ka95_mathcomp.pdf.

[28] Kaltofen, E., and Koiran, P. Finding small degree factors of multivariate supersparse (lacunary) polynomials over algebraic number fields. In *ISSAC MMVI Proc. 2006 Internat. Symp. Symbolic Algebraic Comput.* (New York, N. Y., 2006), J.-G. Dumas, Ed., ACM Press, pp. 162–168. URL: EKbib/06/KaKoi06.pdf.

[29] Kaltofen, E., Krishnamoorthy, M. S., and Saunders, B. D. Fast parallel computation of Hermite and Smith forms of polynomial matrices. *SIAM J. Alg. Discrete Math. 8* (1987), 683–690. URL: EKbib/87/KKS87.pdf.

[30] Kaltofen, E., Krishnamoorthy, M. S., and Saunders, B. D. Parallel algorithms for matrix normal forms. *Linear Algebra and Applications 136* (1990), 189–208. URL: EKbib/90/KKS90.pdf.

[31] Kaltofen, E., and Lakshman Yagati. Improved sparse multivariate polynomial interpolation algorithms. In *Symbolic Algebraic Comput. Internat. Symp. ISSAC '88 Proc.* (Heidelberg, Germany, 1988), P. Gianni, Ed., vol. 358 of *Lect. Notes Comput. Sci.*, Springer Verlag, pp. 467–474. URL: EKbib/88/KaLa88.pdf.

[32] Kaltofen, E., and Lee, W. Early termination in sparse interpolation algorithms. *J. Symbolic Comput. 36*, 3–4 (2003), 365–400. Special issue Internat. Symp. Symbolic Algebraic Comput. (ISSAC 2002). Guest editors: M. Giusti & L. M. Pardo. URL: EKbib/03/KL03.pdf.

[33] Kaltofen, E., Li, B., Yang, Z., and Zhi, L. Exact certification of global optimality of approximate factorizations via rationalizing sums-of-squares with floating point scalars. In *ISSAC 2008* (New York, N. Y., 2008), D. Jeffrey, Ed., ACM Press, pp. 155–163. URL: EKbib/08/KLYZ08.pdf.

[34] Kaltofen, E., and Lobo, A. Distributed matrix-free solution of large sparse linear systems over finite fields. In *Proc. High Performance Computing '96* (San Diego, CA, 1996), A. M. Tentner, Ed., Society for Computer Simulation, Simulation Councils, Inc., pp. 244–247. Journal version in [35]. URL: EKbib/96/KaLo96_hpc.pdf.

[35] Kaltofen, E., and Lobo, A. Distributed matrix-free solution of large sparse linear systems over finite fields. *Algorithmica 24*, 3–4 (July–Aug. 1999), 331–348. Special Issue on "Coarse Grained Parallel Algorithms". URL: EKbib/99/KaLo99.pdf.

[36] Kaltofen, E., May, J., Yang, Z., and Zhi, L. Approximate factorization of multivariate polynomials using singular value decomposition. *J. Symbolic Comput. 43*, 5 (2008), 359–376. URL:

EKbib/07/KMYZ07.pdf.

[37] Kaltofen, E., and Pan, V. Processor efficient parallel solution of linear systems over an abstract field. In *Proc. SPAA '91 3rd Ann. ACM Symp. Parallel Algor. Architecture* (New York, N.Y., 1991), ACM Press, pp. 180–191. URL: EKbib/91/KaPa91.pdf.

[38] Kaltofen, E., and Pan, V. Processor-efficient parallel solution of linear systems II: the positive characteristic and singular cases. In *Proc. 33rd Annual Symp. Foundations of Comp. Sci.* (Los Alamitos, California, 1992), IEEE Computer Society Press, pp. 714–723. URL: EKbib/92/KaPa92.pdf.

[39] Kaltofen, E., and Pan, V. Parallel solution of Toeplitz and Toeplitz-like linear systems over fields of small positive characteristic. In *Proc. First Internat. Symp. Parallel Symbolic Comput. PASCO '94* (Singapore, 1994), H. Hong, Ed., World Scientific Publishing Co., pp. 225–233. URL: EKbib/94/KaPa94.pdf.

[40] Kaltofen, E., and Trager, B. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *J. Symbolic Comput. 9*, 3 (1990), 301–320. URL: EKbib/90/KaTr90.pdf.

[41] Kaltofen, E., and Yang, Z. On exact and approximate interpolation of sparse rational functions. In *ISSAC 2007 Proc. 2007 Internat. Symp. Symbolic Algebraic Comput.* (New York, N. Y., 2007), C. W. Brown, Ed., ACM Press, pp. 203–210. URL: EKbib/07/KaYa07.pdf.

[42] Kaltofen, E., and Yuhasz, G. On the matrix Berlekamp-Massey algorithm, Dec. 2006. Manuscript, 29 pages. Submitted.

[43] Kaltofen, E. L., Li, B., Yang, Z., and Zhi, L. Exact certification in global polynomial optimization via sums-of-squares of rational functions with rational coefficients, Jan. 2009. Accepted for publication in J. Symbolic Comput. URL: EKbib/09/KLYZ09.pdf.

[44] Kaltofen, E. L., and Nehring, M. Supersparse black box rational function interpolation, Jan. 2010. Manuscript, 23 pages.

[45] Lobo, A. A. *Matrix-Free Linear System Solving and Applications to Symbolic Computation.* PhD thesis, Rensselaer Polytechnic Instit., Troy, New York, Dec. 1995.

[46] Massey, J. L. Shift-register synthesis and BCH decoding. *IEEE Trans. Inf. Theory* IT-*15* (1969), 122–127.

[47] Miller, G. L., Ramachandran, V., and Kaltofen, E. Efficient parallel evaluation of straight-line code and arithmetic circuits. *SIAM J. Comput. 17*, 4 (1988), 687–695. URL: EKbib/88/MRK88.pdf.

[48] Pohlig, C. P., and Hellman, M. E. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Trans. Inf. Theory* IT-*24* (1978), 106–110.

[49] Samadani, M., and Kaltofen, E. Prediction based task scheduling in distributed computing. In *Proc. 14th Annual ACM Symp. Principles Distrib. Comput.* (New York, N. Y., 1995), ACM Press, p. 261. Brief announcement of [51, 50].

[50] Samadani, M., and Kaltofen, E. On distributed scheduling using load prediction from past information. Unpublished paper, 1996.

[51] Samadani, M., and Kaltofen, E. Prediction based task scheduling in distributed computing. In *Languages, Compilers and Run-Time Systems for Scalable Computers* (Boston, 1996), B. K. Szymanski and B. Sinharoy, Eds., Kluwer Academic Publ., pp. 317–320. Poster session paper of [50]. URL: EKbib/95/SaKa95_poster.ps.gz.

[52] Stein, W., et al. SAGE: Open Source mathematics software. Web page, Feb. 2008. URL http://www.sagemath.org.

[53] Wiedemann, D. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory* IT-*32* (1986), 54–62.

[54] Zippel, R. Interpolating polynomials from their values. *J. Symbolic Comput. 9*, 3 (1990), 375–403.

[55] Zippel, R. E., Ed. *Proc. 2nd Internat. Workshop on Computer Algebra and Parallelism* (Heidelberg, Germany, 1992), vol. 584 of *Lect. Notes Comput. Sci.*, Springer Verlag.