Design Perspectives for Exact Linear Algebra Libraries

Clément PERNET

UW Number Theory and computation seminar,
November 13, 2008

Introduction

Exact linear algebra:

- over $\mathbb{Z}, \mathbb{Q}, \mathbb{Z}_p$, GF($p^k$).
- `matrix-multiply, solve, rank, det, echelon, charpoly`
- dense, sparse, blackbox matrices

Introduction

Exact linear algebra:

- over $\mathbb{Z}, \mathbb{Q}, \mathbb{Z}_p, \mathrm{GF}(p^k)$.
- `matrix-multiply, solve, rank, det, echelon, charpoly`
- dense, sparse, blackbox matrices

Growing applicative demand

- Number Theory: computing modular forms,
- Crypto: NFS, Groebner bases, ...
- Graph Theory: closure, isomorphism, ...
- High precision approximate linear algebra
- ... (*Mathematics is the art of reducing any problem to linear algebra* [W Stein])

Introduction

Problems are getting more specifics and diverse:

- Specialized arithmetic for GF(2,3,5...),
- Sparse elimination with high rank,

Introduction

Problems are getting more specifics and diverse:

- Specialized arithmetic for GF(2,3,5...),
- Sparse elimination with high rank,

New algorithms

- Asymptoticly faster
- Impact on practical implementations

Introduction

Problems are getting more specifics and diverse:

- Specialized arithmetic for GF(2,3,5...),
- Sparse elimination with high rank,

New algorithms

- Asymptoticly faster
- Impact on practical implementations

New computing architecture framework

- GPU, GPU/CPU
- Parallelism: multi-core vs. grid vs. P2P

Introduction

Existing libraries
fflas-ffpack
M4RI
IML
LinBox

Perspectives
Parallelism and GPU computing
Design considerations
Algorithmic perspectives
GPU computing
Experimentations
Organization of the libraries development
Proposed overall framework
Perspectives

Introduction

A common roof Sage

- interconnect components, small libraries
- easier access for the end user
- has state of the art proper implementations for some linear algebra computations
- community

Introduction

A common roof Sage

- interconnect components, small libraries
- easier access for the end user
- has state of the art proper implementations for some linear algebra computations
- community

Modularity of libraries

- Role of kernel for numerous computations
- Common mistake:

linalg is fun and easy: let's reinvent the wheel!.

Outline

- Existing libraries
  - fflas-ffpack
  - M4RI
  - IML
  - LinBox
- Perspectives
  - Parallelism and GPU computing
    - Design considerations
    - Algorithmic perspectives
    - GPU computing
    - Experimentations
  - Organization of the libraries development
    - Proposed overall framework
    - Perspectives

Outline

Existing libraries
  fflas-ffpack
  M4RI
  IML
  LinBox

Perspectives
  Parallelism and GPU computing
    Design considerations
    Algorithmic perspectives
    GPU computing
    Experimentations
  Organization of the libraries development
    Proposed overall framework
    Perspectives

Outline

Existing libraries
    fflas-ffpack
    M4RI
    IML
    LinBox

Perspectives
    Parallelism and GPU computing
        Design considerations
        Algorithmic perspectives
        GPU computing
        Experimentations
    Organization of the libraries development
        Proposed overall framework
        Perspectives

Dense computations: `FFLAS-FFPACK`

Building block:

matrix multiplication over word-size finite field

Principle:

- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)

Dense computations: `FFLAS-FFPACK`

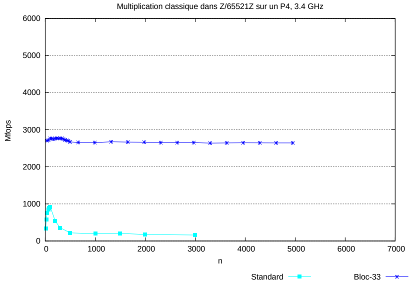
Building block:

matrix multiplication over word-size finite field

Principle:

- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)
- cache tuning

⇒rely on the existing BLAS



Dense computations: `FFLAS-FFPACK`

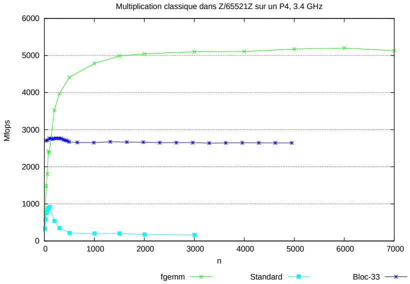
Building block:

matrix multiplication over word-size finite field

Principle:

- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)
- cache tuning

⇒rely on the existing BLAS



Dense computations: `FFLAS-FFPACK`

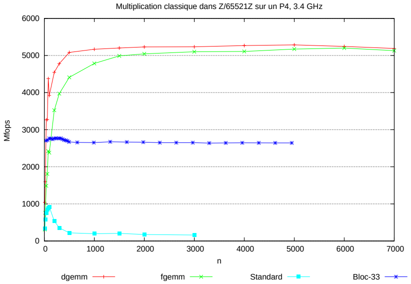
Building block:

matrix multiplication over word-size finite field

Principle:

- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)
- cache tuning

⇒rely on the existing BLAS



Dense computations: `FFLAS-FFPACK`

Building block:

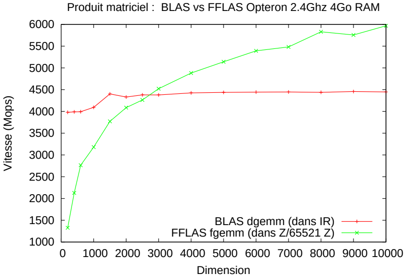
matrix multiplication over word-size finite field

Principle:

- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)
- cache tuning

⇒rely on the existing BLAS

- Sub-cubic algorithm (Winograd)



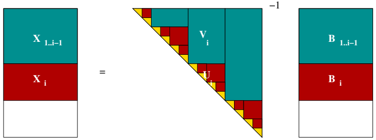
Design of other dense routines

- Reduction to matrix multiplication
- Bounds for delayed modular reductions.

Design of other dense routines

- Reduction to matrix multiplication
- Bounds for delayed modular reductions.

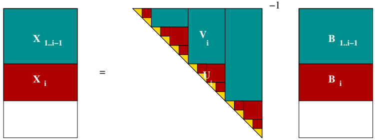
⇒Block algorithm with multiple cascade structures



Design of other dense routines

- Reduction to matrix multiplication
- Bounds for delayed modular reductions.

⇒Block algorithm with multiple cascade structures



	n	*1000*	*2000*	*3000*	*5000*	*10 000*
TRSM	$\frac{ftrsm}{dtrsm}$	1,66	1,33	1,24	1,12	1,01
LQUP	$\frac{lqup}{dgetrf}$	2,00	1,56	1,43	1,18	1,07
INVERSE	$\frac{inverse}{dgetrf+dgetri}$	1.62	1.32	1.15	0.86	0.76

Characteristic polynomial

Fact

$\mathcal{O}(n^\omega)$ *Las Vegas probabilistic algorithm for the computation of the characteristic polynomial over a Field.*

Characteristic polynomial

Fact

$\mathcal{O}(n^\omega)$ *Las Vegas probabilistic algorithm for the computation of the characteristic polynomial over a Field.*

Practical algorithm :

Characteristic polynomial

Fact

$\mathcal{O}(n^\omega)$ Las Vegas probabilistic algorithm for the computation of the characteristic polynomial over a Field.

Practical algorithm :

n	500	5000	15 000
`LinBox`	0.91s	4m44s	2h20m
`magma-2.13`	1.27s	15m32s	7h28m

Characteristic polynomial

Fact

$\mathcal{O}(n^\omega)$ Las Vegas probabilistic algorithm for the computation of the characteristic polynomial over a Field.

Practical algorithm :

n	500	5000	15 000
`LinBox`	0.91s	4m44s	2h20m
`magma-2.13`	1.27s	15m32s	7h28m

- Frobenius normal form as well
- Transformation in $\mathcal{O}(n^\omega \log\log n)$

Outline

Existing libraries
- fflas-ffpack
- M4RI
- IML
- LinBox

Perspectives
- Parallelism and GPU computing
  - Design considerations
  - Algorithmic perspectives
  - GPU computing
  - Experimentations
- Organization of the libraries development
  - Proposed overall framework
  - Perspectives

M4RI: Dense linear algebra over GF(2)

[Albrecht, Bard & Al.]

Provides:

- Matrix multiplication
- Echelon Form
- System solution

M4RI: Dense linear algebra over GF(2)

[Albrecht, Bard & Al.]

Provides:

- Matrix multiplication
- Echelon Form
- System solution

Using:

- Packed representation of elements: `long long` $\equiv \mathrm{GF}(2)^{64}$
- Greasing technique: multiplication tables, and Gray codes
- SSE2 support and cache friendliness
- sub-cubic matrix arithmetic

Performance

n	magma-2.14	GAP	M4RI
10000	1.892s	6.13s	1.504s
16384	7.720s	25.04s	6.074s
20000	13.21s	-	10.72s
32000	53.67s	-	43.19s

Matrix multiplication on a Core 2 Duo 2.33Ghz

n	magma-2.14	GAP	M4RI
10000	3.283s	23.573s	2.509s
16384	11.204s	-	10.741s
20000	16.911s	-	19.776s
32000	57.761s	-	86.071s
64000	355.477s	-	640.742s

Matrix inversion on a Opteron 2.6Ghz,

Outline

Existing libraries

- fflas-ffpack
- M4RI
- IML
- LinBox

Perspectives

- Parallelism and GPU computing
  - Design considerations
  - Algorithmic perspectives
  - GPU computing
  - Experimentations
- Organization of the libraries development
  - Proposed overall framework
  - Perspectives

IML: Integer Matrix Library

[Chen, Storjohann]

Provides:

- Dense linear system solver over $\mathbb{Q}$

Using:

- BLAS based dense linear algebra over finite fields
- Efficient Dixon Lifting, system solver
- LLL reduction

Outline

Existing libraries

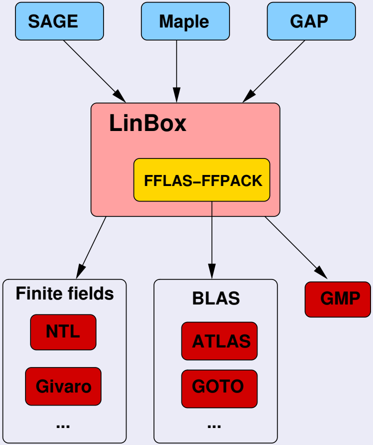
- fflas-ffpack
- M4RI
- IML
- LinBox

Perspectives

- Parallelism and GPU computing
  - Design considerations
  - Algorithmic perspectives
  - GPU computing
  - Experimentations
- Organization of the libraries development
  - Proposed overall framework
  - Perspectives

LinBox

A generic middleware



LinBox-1.0

Solutions

- rank
- det
- minpoly
- charpoly
- system solve
- positive definiteness

LinBox-1.0

Solutions

- rank
- det
- minpoly
- charpoly
- system solve
- positive definiteness

Domains of computation

- Finite fields
- $\mathbb{Z}$

LinBox-1.0

Solutions

- rank
- det
- minpoly
- charpoly
- system solve
- positive definiteness

Domains of computation

- Finite fields
- $\mathbb{Z}$

Matrices

- Sparse, structured
- Dense

LinBox-1.0

Solutions

- rank
- det
- minpoly
- charpoly
- system solve
- positive definiteness

Domains of computation

- Finite fields
- $\mathbb{Z}$

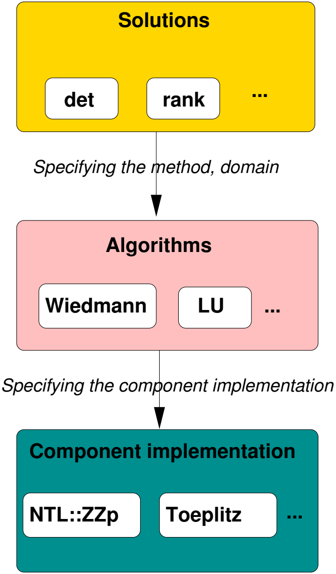
Matrices

- Sparse, structured
- Dense

Genericity:

- Domains of computation
- Matrices, vectors
- Algorithms (containers and iterators)

Structure of the library



Solutions

det

rank

...

Specifying the method, domain

Algorithms

Wiedmann

LU

...

Specifying the component implementation

Component implementation

NTL::ZZp

Toeplitz

...

BlackBox computations

BlackBox computations

$A \in K^{n \times m}$

$v \in K^{m}$

$Av \in K^{n}$

Goal: computation with very large sparse or structured matrices.

- No explicit representation of the matrix,
- Only operation: application of a vector

BlackBox computations

Goal: computation with very large sparse or structured matrices.

- No explicit representation of the matrix,
- Only operation: application of a vector
- Efficient algorithms
- Efficient preconditionners: Toeplitz, Hankel, Butterfly, ...

Block projection algorithms

- Wiedemann algorithm: scalar projections of A^i for $i = 1..2d$
- Block Wiedemann: $k \times k$ dense projections of A^i for $i = 1..2d/k$

⇒Balance efficiency between BlackBox and dense compations

Outline

Existing libraries
  - fflas-ffpack
  - M4RI
  - IML
  - LinBox

Perspectives
  - Parallelism and GPU computing
    - Design considerations
    - Algorithmic perspectives
    - GPU computing
    - Experimentations
  - Organization of the libraries development
    - Proposed overall framework
    - Perspectives

Design of linear algebra libraries

Two main goals:

Pursue the quest for efficiency

- New architecture infrastructures: parallelism and GPU computing
- Different approaches to tuning: BLAS, low level tuning, ...
- Algorithmic improvements

Design of linear algebra libraries

Two main goals:

Pursue the quest for efficiency

- New architecture infrastructures: parallelism and GPU computing
- Different approaches to tuning: BLAS, low level tuning, ...
- Algorithmic improvements

Time to reorganize the libraries

- Separate basic linear algebra, to higher level algs
- Specialize low level linear algebra on some domain of interests
- Keep advanced algorithms generic

Outline

Existing libraries
- fflas-ffpack
- M4RI
- IML
- LinBox

Perspectives
- Parallelism and GPU computing
  - Design considerations
  - Algorithmic perspectives
  - GPU computing
  - Experimentations
- Organization of the libraries development
  - Proposed overall framework
  - Perspectives

Data Containers/Iterators

Distinction between computation and access to the data:

Example

Iterates $(u^T A^i v)_{i=1..k}$ *used for system resolution can be*

- *precomputed and stored*
- *computed on the fly*
- *computed in parallel*

Data Containers/Iterators

Distinction between computation and access to the data:

Example

Iterates $(u^T A^i v)_{i=1..k}$ *used for system resolution can be*

- *precomputed and stored*
- *computed on the fly*
- *computed in parallel*

Solution: solver defined using generic iterators, independently from the method to compute the data

Example: A parallel data flow iterator

```
 const iterator& iterator::operator++(){
    if (++current>launched) {
        ...
        for (int i=0; i<n; ++i)
            Fork<launch>(i,...);
        launched += n;
        }
    return *this;
}
 const value_type& iterator::operator*(){
    return _d[current].read();
}
```

Existing containers/iterators

- Scalar projections: $(v^T A^i u)_{i=1..k}$
  $\Rightarrow$Wiedemann's algorithm

Existing containers/iterators

- Scalar projections: $(v^T A^i u)_{i=1..k}$
  $\Rightarrow$Wiedemann's algorithm
- Block projections: $(Av_i)_{i=1..k}$
  $\Rightarrow$Block Wiedemann algorithm

Existing containers/iterators

- Scalar projections: $(v^T A^i u)_{i=1..k}$
  $\Rightarrow$Wiedemann's algorithm
- Block projections: $(Av_i)_{i=1..k}$
  $\Rightarrow$Block Wiedemann algorithm
- Modular homomorphic imaging:
  $$(\texttt{Algorithm}(A \mod p_i))_{i=1..k}$$
  $\Rightarrow$Chinese Remainder Algorithm

Existing containers/iterators

- Scalar projections: $(v^T A^i u)_{i=1..k}$
  $\Rightarrow$Wiedemann's algorithm
- Block projections: $(Av_i)_{i=1..k}$
  $\Rightarrow$Block Wiedemann algorithm
- Modular homomorphic imaging: $(\texttt{Algorithm}(A \mod p_i))_{i=1..k}$
  $\Rightarrow$Chinese Remainder Algorithm

$\Rightarrow$no modifications to the high level algorithms for the parallelization.

Parallelization tools

Until now, few parallelization:

- attempts with MPI, and POSIX threads
- Higher level systems: Athapascan-1, KAAPI
  ⇒Full design compatibility
  ⇒Provides efficient schedulers; work stealing abilities

Example: rank computations

[Dumas & urbanska]

- parallel block Wiedemann algorithm: $[u_1, .., u_k]^T (GG^T) u_i, i = 1..k$
  $\Rightarrow$Only $\frac{rank(G)}{k}$ iterations
- combined with sigma basis algorithm.

Example: rank computations

[Dumas & urbanska]

- parallel block Wiedemann algorithm: $[u_1, .., u_k]^T (GG^T) u_i, i = 1..k$
  $\Rightarrow$Only $\frac{rank(G)}{k}$ iterations
- combined with sigma basis algorithm.

matrix	n	m	rank
GL7d17	1,548,650	955,128	626,910
GL7d20	1,437,547	1,911,130	877,562
GL7d21	822,922	1,437,547	559,985

Example: rank computations

[Dumas & urbanska]

- parallel block Wiedemann algorithm: $[u_1, .., u_k]^T (GG^T) u_i, i = 1..k$
  $\Rightarrow$Only $\frac{rank(G)}{k}$ iterations
- combined with sigma basis algorithm.

matrix	n	m	rank
GL7d17	1,548,650	955,128	626,910
GL7d20	1,437,547	1,911,130	877,562
GL7d21	822,922	1,437,547	559,985

Timings estimations [in days]

matrix	T_{iter} [min]	T_{seq}	T_{par} (50)	T_{par} (50, ET)
GL7d17	0.46875	621.8	12.4	8.16
GL7d20	0.68182	1361.31	27.2272	16.6214
GL7d21	0.35714	408.196	8.1644	5.5559

Outline

Existing libraries

- fflas-ffpack
- M4RI
- IML
- LinBox

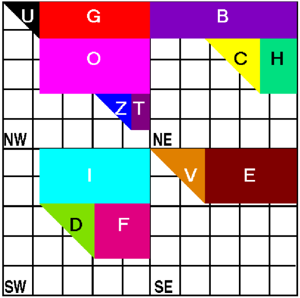
Perspectives

- Parallelism and GPU computing
  - Design considerations
  - Algorithmic perspectives
  - GPU computing
  - Experimentations
- Organization of the libraries development
  - Proposed overall framework
  - Perspectives

TURBO triangular elimination

[Roch & Dumas 02]: recursive block algorithm for triangularization

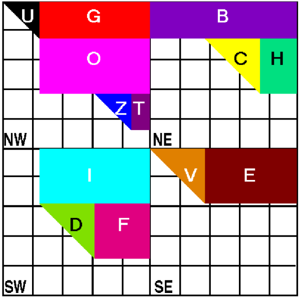
- divide both rows and columns
  ⇒Better memory management
  ⇒Enables to use recursive data structures

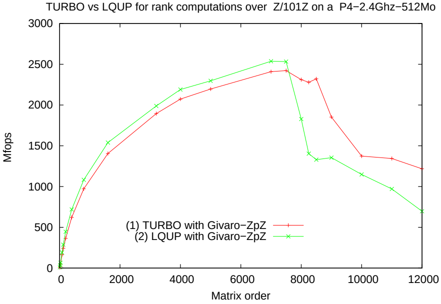


TURBO triangular elimination

[Roch & Dumas 02]: recursive block algorithm for triangularization

- divide both rows and columns
  ⇒Better memory management
  ⇒Enables to use recursive data structures

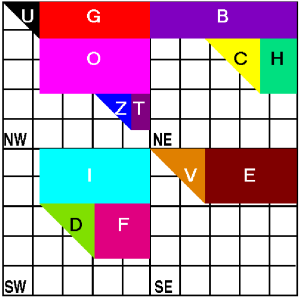


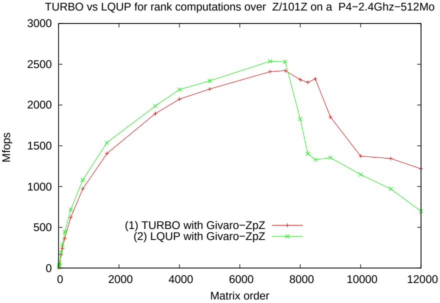


TURBO triangular elimination

[Roch & Dumas 02]: recursive block algorithm for triangularization

- divide both rows and columns
  ⇒Better memory management
  ⇒Enables to use recursive data structures
- 5 recursive calls (U, V, C, D, Z), including 2 being parallel (C, D)

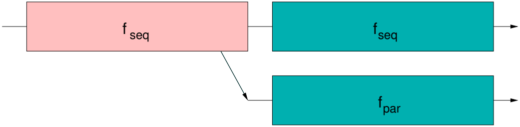




Principle of Workstealing

[Arora, Blumofe, Plaxton01], [Acar, Blelloche, Blumofe02]

- 2 algorithms to complete a task f: f_{seq} and f_{par}
- When a processor becomes idle, `ExtractPar` *steals* the work to f_{seq}.



Application to mutiple triangular system solving

$$\texttt{TRSM}: \text{Compute } \begin{bmatrix} U_1 & U_2 \\ & U_3 \end{bmatrix}^{-1} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$$

$$\begin{aligned} X_2 &= \texttt{TRSM}(U_3, B_2) \\ B_1 &= B_1 - U_2 X_2 \\ X_1 &= \texttt{TRSM}(U_1, B_1) \end{aligned}$$

Application to mutiple triangular system solving

$$\texttt{TRSM}: \text{Compute } \begin{bmatrix} U_1 & U_2 \\ & U_3 \end{bmatrix}^{-1} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$$

$$\begin{aligned} X_2 &= \texttt{TRSM}(U_3, B_2) \\ B_1 &= B_1 - U_2 X_2 \\ X_1 &= \texttt{TRSM}(U_1, B_1) \end{aligned}$$

f_{seq}

$\texttt{TRSM}(U, B)$

$\Rightarrow T_1 = n^3, \ T_\infty = \mathcal{O}(n)$

Application to mutiple triangular system solving

$$\texttt{TRSM}: \text{Compute} \begin{bmatrix} U_1 & U_2 \\ & U_3 \end{bmatrix}^{-1} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$$

$$\begin{aligned} X_2 &= \texttt{TRSM}(U_3, B_2) \\ B_1 &= B_1 - U_2X_2 \\ X_1 &= \texttt{TRSM}(U_1, B_1) \end{aligned}$$

f_{seq}

$\texttt{TRSM}(U, B)$

$\Rightarrow T_1 = n^3,\ T_\infty = \mathcal{O}(n)$

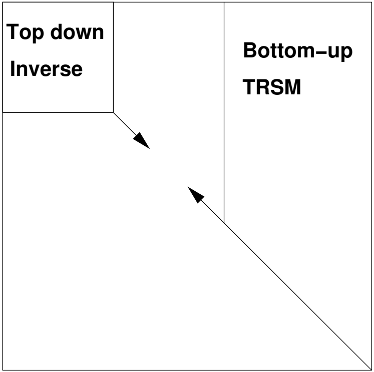
f_{par}

Compute $V = U^{-1}$;

$\texttt{TRMM}(V, B)$;

$\Rightarrow T_1 = \frac{4}{3}n^3,\ T_\infty = \mathcal{O}(\log n)$

Application to multiple triangular system solving



When sequential `TRSM` and parallel `Inverse` join:
Compute $X_1 = A_1^{-1} B_1$ in parallel (`TRMM`).

Multi-adic lifting

Solving $Ax = b$ over $\mathbb{Z}$

Standard p-adic Lifting [Dixon82]

```
Compute $A^{-1} \mod p$
$r = b$
for $i = 0..n$ do
    $x_i = A^{-1} r \mod p$
    $r = (r - Ax_i)/p$
end for
$z = x_0 + px_1 + p^2 x_2 + \cdots + x_n p^n$
$x = \mathtt{RationalReconstruction}(z)$
```

Multi-adic lifting

Solving $Ax = b$ over $\mathbb{Z}$

multi-adic lifting:

```
for all j=1..k do
    Compute $A^{-1} \mod p_j$
    $r = b$
    for $i = 0..n/k$ do
        $x_i = A^{-1} r \mod p_j$
        $r = (r - Ax_i)/p_j$
    end for
    $z_j = x_0 + p_j x_1 + \cdots + p_j^{n/k} x_{n/k}$
end for
$z = \text{ChineseRemainderAlg}((z_j, p_j^{n/k})_{j=1..k})$
$x = \texttt{RatReconst}(z)$
```

Multi-adic lifting

- Used in sequential computation [Chen & Storjohann 05], to balance efficiency between BLAS level 2 and 3

Multi-adic lifting

- Used in sequential computation [Chen & Storjohann 05], to balance efficiency between BLAS level 2 and 3
- Divides a sequential loop into several parallel tasks

Multi-adic lifting

- Used in sequential computation [Chen & Storjohann 05], to balance efficiency between BLAS level 2 and 3
- Divides a sequential loop into several parallel tasks
- Work stealing perspectives...

A history of the GPU technology

- “dumb” framebuffer
- Bitblt: copy interleaved rgb bitmaps quickly
- offloading of 3D computations to the GPU, i.e. z-buffers
- (primitive early) Shaders: small, up to 128 instructions, no branching, etc
- CuDa: C compiler that produces code running on the GPU

Interesting GPUs

- current NVidia: Tesla C870 GPU. 128 thread processors, 1.5 GB dedicated Memory
- Fall 2008 Nvidia: Tesla C1060. 240 thread processors, 4 GB dedicated memory at 102 GB/sec, 90 GFlops Double Precision, 360 GFlops Single Precision
- current ATI: RV770, 800 SPs, 1GB+ dedicated Memory, 1.2TFLOPS single precision, 150GFlops Double Precision
- Intel: Larrabee - A Many-Core x86 Architecture for Visual Computing (Vaporware, 1TFlop Single Precision)

Most of the above will/are conforming to IEEE specs. In comparison: Intel high end Core2 Quad: 100 GFlops Single Precision

GPU Alternatives

- FPGAs
- IBM's Cell CPU, especially the second generation
- CPU/GPU combos, i.e. AMD/ATi

GPU: Programming Tools

- CuDABLAS: Easy to hook into existing (numerical software)
- generic CuDA: Write C code, compile it to GPU code (this is NVidia specific)
- OpenCL
- Intel's Secret Sauce

Using CUDA BLAS interface to GPU

```
cublasAlloc(n*n,sizeof(*a),(void**)\&devPtrA);
cublasSetMatrix (n,n,sizeof(*a),a,n,devPtrA,n);

cublasAlloc(n*n,sizeof(*b),(void**)\&devPtrB);
cublasSetMatrix (n,n,sizeof(*b),a,n,devPtrB,n);

cublasAlloc(n*n,sizeof(*c),(void**)\&devPtrC);
cublasSetMatrix (n,n,sizeof(*c),c,n,devPtrC,n);

cublasSgemm (’N’, ’N’, n, n, n, 1.0,
             devPtrA, n,
             devPtrB, n,
             0.0, devPtrC, n);

cublasGetMatrix (n, n, sizeof(*c), devPtrC, n, c, n);

cublasFree(devPtrA);
cublasFree(devPtrB);
cublasFree(devPtrC);
```

FFPACK over GPU's

n	1000	1500	2000	2500
naive	8.0s	32.2s	82.1s	167s
GPU: CUDA	.65s	.97s	1.87s	4.28s
CPU: ATLAS (2cores)	.13s	.43s	1.07s	1.86s

Matrix multiplication over $\mathbb{Z}_{11}$
using BLAS sgemm (32 bits floats)

Outline

- Existing libraries
  - fflas-ffpack
  - M4RI
  - IML
  - LinBox
- Perspectives
  - Parallelism and GPU computing
    - Design considerations
    - Algorithmic perspectives
    - GPU computing
    - Experimentations
  - Organization of the libraries development
    - Proposed overall framework
    - Perspectives

Proposed overall framework

Split libraries into dedicated roles:

Basic operations over specific matrix classes and domains of computation

FFPACK: dense over $\mathbb{Z}_p, p \leq 2^{26}$, $GF(p^k), p^k < 2^8$

M4RI: dense (, sparse ?) over $GF(2)$

IML (LAMPIR ?): dense over $\mathbb{Z}, \mathbb{Q}$

Polynomial linear algebra: dense over $K[X]$ where $K = \mathbb{Z}_p, \mathrm{GF}(p^k), \mathbb{Q}, \mathbb{Z}$

Proposed overall framework

Split libraries into dedicated roles:

Basic operations over specific matrix classes and domains of computation

FFPACK: dense over $\mathbb{Z}_p, p \leq 2^{26}$, $GF(p^k), p^k < 2^8$

M4RI: dense (, sparse ?) over $GF(2)$

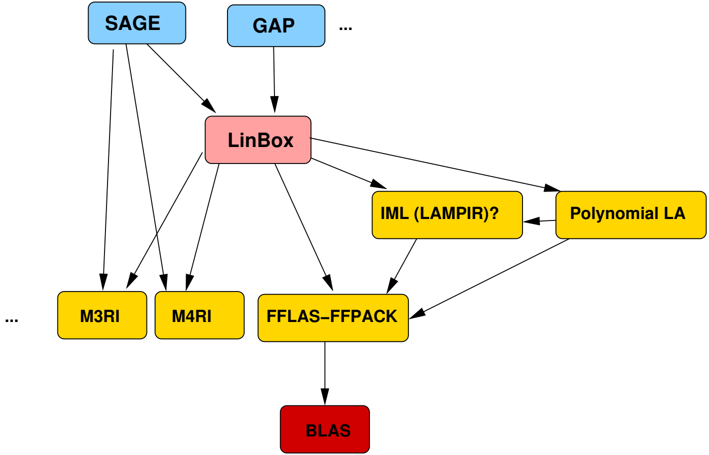
IML (LAMPIR ?): dense over $\mathbb{Z}, \mathbb{Q}$

Polynomial linear algebra: dense over $K[X]$ where $K = \mathbb{Z}_p, \mathrm{GF}(p^k), \mathbb{Q}, \mathbb{Z}$

more advanced algorithms over generic domains

- LinBox, Sage

Proposed overall framework



SAGE

GAP

...

LinBox

IML (LAMPIR)?

Polynomial LA

...

M3RI

M4RI

FFLAS–FFPACK

BLAS

Perspectives

FFPACK:

- Integration of compressed modular representations for small finite fields.
- Design for GPU and multicore support

IML (LAMPIR?):

- Merge LinBox and IML rational solvers.
- Use LinBox iterators abstraction to enable parallelism
- Small dimension / big integers.
  - implement a lot of heuristics
  - automated composition of algorithms

Perspectives

M4RI:

- Sub-cubic matrix arithmetic (echelon, inversion, ...)
- Sparse $GF(2)$

Perspectives

M4RI:

- Sub-cubic matrix arithmetic (echelon, inversion, ...)
- Sparse $GF(2)$

M3RI, M5RI, ..., MiRI: [Boothby, Bradshaw]

- Encouraging first experiments
- Factorize out the common design
- Automated generation of code given high level specifications [FLAME project]

Perspectives

M4RI:

- Sub-cubic matrix arithmetic (echelon, inversion, ...)
- Sparse $GF(2)$

M3RI, M5RI, ..., M*i*RI: [Boothby, Bradshaw]

- Encouraging first experiments
- Factorize out the common design
- Automated generation of code given high level specifications [FLAME project]

LinBox:

- Sparse elimination algorithms: more solutions
- BlackBoxes: more recent algorithms, more hybrid BB/dense methods