# Computing exactly with unsafe resources: fault tolerant exact linear algebra and cloud computing

Majid KHONJI and Clément PERNET,
joint work with
Jean-Louis ROCH and Thomas STALINSKY

INRIA-MOAIS, Grenoble Univ. France

SAGE Days 16,
June 23, 2009

# Outline

# Outline






Inroduction
Distributed and Cloud computing
Fault tolerance
Parallel Exact linear algebra
Chinese Remainder and Error correcting code
Principle
Correction
Overview of the computation scheme
Further improvements
Ordering, sign
Early termination
Updated computation scheme
Application to computing Determinants
Softwares
Live demo
Perspectives

# High performance mathematics computing

Numerous application areas

**Number theory:**
- computing tables of elliptic curves, modular forms,
- testing of conjectures,

**Crypto:**
- Algebraic attacks
- Search for big primes

**Graph theory:** testing conjectures (graph isomorph.,...)

**Your research pb here!** ....

# Parallel computations

## (Re-)birth of parallel computing

Personal computers ⇒ multi/many cores

- End of the race for CPU clock-frequency
- Multi-core: 2, 4, 6, ...
- Many-cores: near future, already in GPU's
- multi-GPU

# Parallel computations

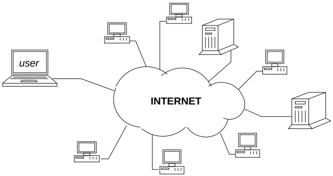
(Re-)birth of parallel computing

Personal computers ⇒multi/many cores

- End of the race for CPU clock-frequency
- Multi-core: 2, 4, 6, ...
- Many-cores: near future, already in GPU's
- multi-GPU

HPC ⇒Global computing

- servers
- grids, clusters,
- volunteer computing, Peer to Peer
- cloud computing




⇒Shift towards massively parallel computations

# Volunteer and Peer to peer computing

Well known successful projects:

- Mersenne Prime search
- SETI@Home
- Folding@Home/BOINC :
  ⇒1 Petaflops (670 000 PS3s) in 2007
- ...

⇒What level of trust?

# Cloud Computing

[Above the clouds, a Berkeley view on Cloud Computing, Feb 09]

Difficult agreement on the definition

Based on

- Software As A Service
- Computing centers providing ressources in a pay as you go fashion

# Cloud Computing

[Above the clouds, a Berkeley view on Cloud Computing, Feb 09]

Difficult agreement on the definition

Based on

- Software As A Service
- Computing centers providing ressources in a pay as you go fashion

*The interesting thing about Cloud Computing is that we've redefined Cloud Computing to include everything that we already do... I don't understand what we would do differently in the light of Cloud Computing other than change the wording of some of our ads.*

*Larry Ellison*

*It's stupidity, It's worse than stupidity: it's a marketing hype campaign.*

*R. Stallman*

# Cloud Computing

Interest:

- illusion of infinite computing ressources
- no up-front commitment by cloud users
- processor by the hour, storage by the day

# Cloud Computing

Interest:

- illusion of infinite computing ressources
- no up-front commitment by cloud users
- processor by the hour, storage by the day

$\Rightarrow$for massively parallel computations:
New *cost associativity* :

100's of computers for 1h $\equiv$ 1 computer for 100h

$\Rightarrow$What level of trust

# Fault tolerance

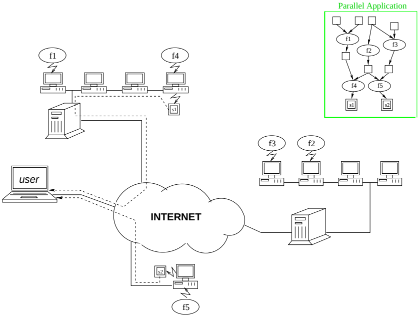
Need to deal with faults of different types

- Fail stop (crash failures)
- Network congestion
- Malicious attack

⇒Model of the Byzantine failure (not consistently failing)

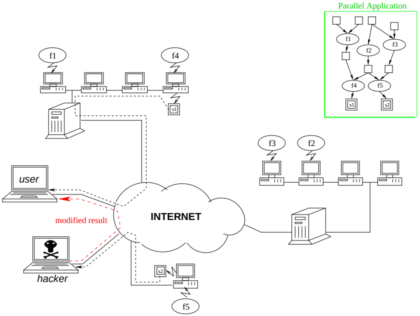
# Global computing

- Grid and P2P: Transparent allocation of the resources to authenticated users
  *scheduling supports ressource connections / disconnections*



# Global computing and task forgery

- Grid and P2P: Transparent allocation of the resources to authenticated users
  *scheduling supports ressource connections / disconnections*




- Yet a task can be **forged** $\iff f(\text{ input }) \neq \hat{f}(\text{ input })$
- forgery can affect *many* tasks [e.g. patched client in SETI@home]

# State-of-the-Art in Result Certification

- Mainly target programs **P** composed of independent tasks
- General approach: *Replication-based*
  - Voting [*e.g.* BOINC, SETI@home]
  - Spot-checking [Germain-Playez03, based on Wald test]
  - checkpoint and restart
    $\Rightarrow$30' to 1h on large clusters!
  - Blacklisting, Credibility-based fault-tolerance [Sarmenta03]
  - Partial execution on reliable resources [Gao-Malewicz04]

# State-of-the-Art in Result Certification

- Mainly target programs **P** composed of independent tasks
- General approach: *Replication-based*
  - Voting [*e.g.* BOINC, SETI@home]
  - Spot-checking [Germain-Playez03, based on Wald test]
  - checkpoint and restart
    $\Rightarrow$30' to 1h on large clusters!
  - Blacklisting, Credibility-based fault-tolerance [Sarmenta03]
  - Partial execution on reliable resources [Gao-Malewicz04]
- Specific approach: *check post-condition* on the results
  - Eg: Sorting $\mathcal{O}(n \log n)$ – Simple-Checker $\mathcal{O}(n)$ [Blum97]
  - **The most efficient approach when possible!**

Yet, no guarantee of result correction without hypothesis on the attack

# Approach: Massive versus localized attacks

- In practice:
  - DAG of tasks, highly parallel
  - for most executions, only few or none forged tasks!
    $\hookrightarrow$ full replication useless and too expensive

- Yet, **no blind trust**:
  - large scale falsifications are possible $\hookrightarrow$ can be detected by trustable verification of randomly selected tasks:
    Extended Monte-Carlo Certification **EMCT**
    [Krings&al 06]
  - few falsifications are possible $\hookrightarrow$ can be efficiently corrected by Algorithm-based fault tolerance (**ABFT**)
    [Beckman 93, Plank&al 97, Saha 2006]

# Approach: Massive versus localized attacks

- In practice:
  - DAG of tasks, highly parallel
  - for most executions, only few or none forged tasks!
    $\hookrightarrow$ full replication useless and too expensive

- Yet, **no blind trust**:
  - large scale falsifications are possible $\hookrightarrow$ can be detected by trustable verification of randomly selected tasks:
    Extended Monte-Carlo Certification **EMCT**
    [Krings&al 06]
  - few falsifications are possible $\hookrightarrow$ can be efficiently corrected by Algorithm-based fault tolerance (**ABFT**)
    [Beckman 93, Plank&al 97, Saha 2006]

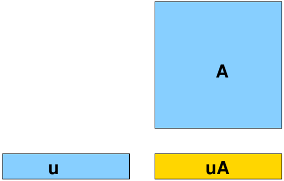
# Algorithmic Based Fault Tolerance

Idea: incorporate the redundancy in the algorithm
$\Rightarrow$ use some specific properties of the problem

# Algorithmic Based Fault Tolerance

**Idea:** incorporate the redundancy in the algorithm
⇒use some specific properties of the problem

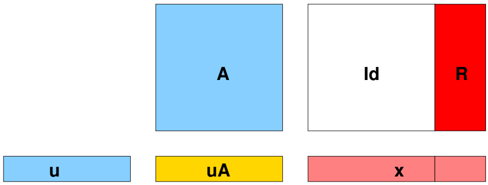
**Example:** Matrix vector product [Dongarra 2006]



# Algorithmic Based Fault Tolerance

**Idea:** incorporate the redundancy in the algorithm
⇒use some specific properties of the problem

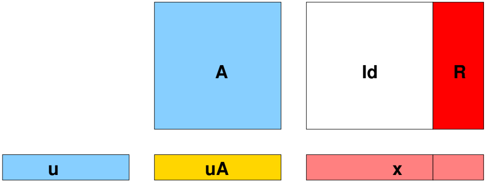
**Example:** Matrix vector product [Dongarra 2006]



# Algorithmic Based Fault Tolerance

**Idea:** incorporate the redundancy in the algorithm
$\Rightarrow$use some specific properties of the problem

**Example:** Matrix vector product [Dongarra 2006]




- precompute the product $B = A \times \begin{bmatrix} I & R \end{bmatrix}$
- compute $x = uB$ in parallel
- decode/correct $x$

# Parallel Exact linear algebra

*Mathematics is the art of reducing any problem to linear algbra*

*W. Stein*

⇒the core to be optimized for a lot of computations

# Parallel Exact linear algebra

*Mathematics is the art of reducing any problem to linear algbra*

*W. Stein*

⇒the core to be optimized for a lot of computations

- Matrix multiplication over GF(p)
- Eliminations: Gauss, Gram-Schmidt (LLL), ...
- Krylov iteration
- Chinese Remainder Algorithm

# Parallel Exact linear algebra

*Mathematics is the art of reducing any problem to linear algbra*

*W. Stein*

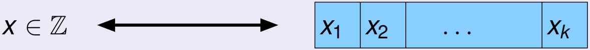
⇒the core to be optimized for a lot of computations

- Matrix multiplication over GF(p)
- Eliminations: Gauss, Gram-Schmidt (LLL), ...
- Krylov iteration
- Chinese Remainder Algorithm

# Outline

# Principle

## Chinese Remainder Theorem




where $p_1 \times \cdots \times p_k > x$ and $x_i = x \mod p_i \; \forall i$

# Principle

## Chinese Remainder Theorem

$x \in \mathbb{Z} \longleftrightarrow$

| | | | | | | |
|---|---|---|---|---|---|---|
| $x_1$ | $x_2$ | $\dots$ | $x_k$ | $x_{k+1}$ | $\dots$ | $x_n$ |

where $p_1 \times \cdots \times p_n > x$ and $x_i = x \mod p_i \; \forall i$

# Principle

## Chinese Remainder Theorem

$x \in \mathbb{Z} \longleftrightarrow$

| | | | | | | |
|---|---|---|---|---|---|---|
| $x_1$ | $x_2$ | $\ldots$ | $x_k$ | $x_{k+1}$ | $\ldots$ | $x_n$ |

where $p_1 \times \cdots \times p_n > x$ and $x_i = x \mod p_i \; \forall i$

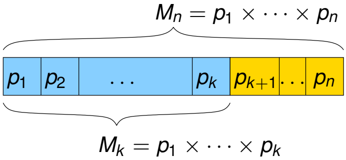
## Definition

*$(n,k)$-code:*
$C = \{(x_1, \ldots, x_n) \in \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_n}$ *s.t. there exists* $x, 0 \leq x < p_1 \ldots p_k$ *and* $x_i = x \mod p_i \; \forall i\}$.

# Principle

## Property

$$X \in C \text{ iff } X < M_k.$$



# Principle

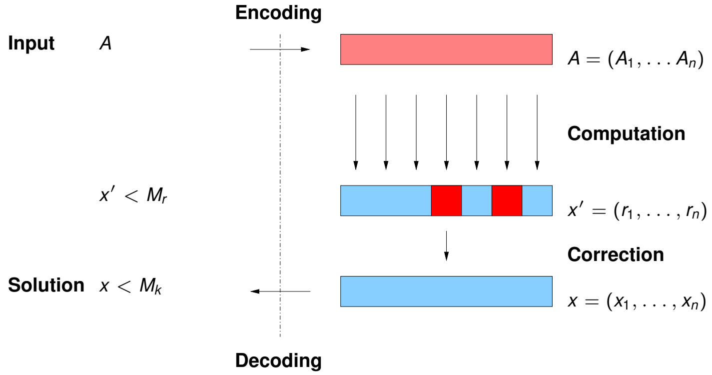
Transmission channel $\equiv$ Computation

# Principle

Noisy Transmission channel $\equiv$ Unsafe Computation

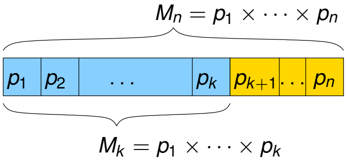
# Principle

Noisy Transmission channel $\equiv$ Unsafe Computation



# Property of the code

Redundancy : $r = n - k$



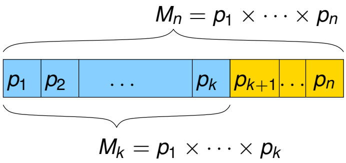

### Detects up to $r$ errors:

For $X' = X + E$ with $X \in C, \|E\| \leq r$,

$$X' > M_k.$$

# Property of the code

Redundancy : $r = n - k$




**Detects up to $r$ errors:**

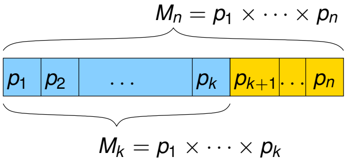
For $X' = X + E$ with $X \in C, \|E\| \leq r$,

$$X' > M_k.$$

- Distance: $r + 1$
- Corrects up to $\lfloor \frac{r}{2} \rfloor$ errors: the closest code word

# Property of the code

Redundancy : $r = n - k$




**Detects up to $r$ errors:**

For $X' = X + E$ with $X \in C, \|E\| \leq r$,

$$X' > M_k.$$

- Distance: $r + 1$
- Corrects up to $\lfloor \frac{r}{2} \rfloor$ errors: the closest code word

# Correction

[Mandelbaum 78]

## CRT

- $\pi = p_1 \times \cdots \times p_n$,
- $\pi_i = \pi / p_i$
- $y_i = \pi_i^{-1} \mod p_i$.

$$x = \sum x_i \pi_i y_i$$

# Correction

[Mandelbaum 78]

### CRT

- $\pi = p_1 \times \cdots \times p_n$,
- $\pi_i = \pi / p_i$
- $y_i = \pi_i^{-1} \mod p_i$.

$$x = \sum x_i \pi_i y_i$$

- Error $e$ of weight $t \leq \left\lfloor \frac{r}{2} \right\rfloor$
- $C$: the product of the $p_i$ of the errors

$$\begin{aligned} x &= x' - e \\ &= x' - \sum_{\text{supp}(e)} x_i \pi_i y_i \\ &= x' - \frac{M_n}{C} B' \end{aligned}$$

for some $0 \leq B' \leq C$

# Correction

## Theorem

*If* $e \leq (n-k)\frac{\log p_{\min} - \log 2}{\log p_{\max} + \log p_{\min}}$

$$\left| \frac{x'}{M_n} - \frac{B'}{C} \right| \leq \frac{1}{C^2}$$

$\Rightarrow$ use continued fractions to approximate $\frac{x'}{M_n}$

$\Rightarrow$ recover $B'/C$ $\quad\Rightarrow e$

# Continued fractions

**Continued fractions:** Every rational can be written as

$$\frac{p}{q} = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \dots}} = [a_0, \dots, a_n]$$

**Convergents:** a truncation $\frac{p_i}{q_i} = [a_0, \dots, a_i]$ for $i \leq n$

# Continued fractions

Continued fractions: Every rational can be written as

$$\frac{p}{q} = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \dots}} = [a_0, \dots, a_n]$$

Convergents: a truncation $\frac{p_i}{q_i} = [a_0, \dots, a_i]$ for $i \leq n$

## Property

$\left|\frac{p}{q} - \frac{p_i}{q_i}\right| \leq \frac{1}{q_i^2}$ *and is decreasing.*

# Continued fractions

Continued fractions: Every rational can be written as

$$\frac{p}{q} = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \dots}} = [a_0, \dots, a_n]$$

Convergents: a truncation $\frac{p_i}{q_i} = [a_0, \dots, a_i]$ for $i \leq n$

## Property

$\left| \frac{p}{q} - \frac{p_i}{q_i} \right| \leq \frac{1}{q_i^2}$ *and is decreasing.*

$\Rightarrow$

- compute iteratively the convergents of $\frac{x'}{M_n}$ until $\left| \frac{x'}{M_n} - \frac{p_i}{q_i} \right| \leq \frac{1}{C^2}$ is satisfied.
- If $M_n \frac{p_i}{q_i} \in \mathbb{Z}$ and $x = x' - e \leq M_k$ $\Rightarrow$ Corrected

# Outline





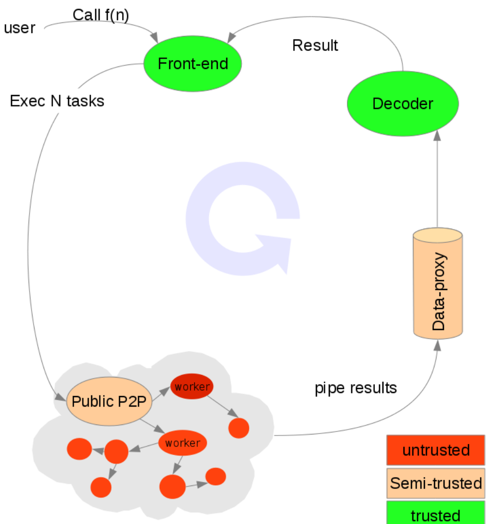
Inroduction
Distributed and Cloud computing
Fault tolerance
Parallel Exact linear algebra
Chinese Remainder and Error correcting code
Principle
Correction
Overview of the computation scheme
Further improvements
Ordering, sign
Early termination
Updated computation scheme
Application to computing Determinants
Softwares
Live demo
Perspectives

# Computation resources

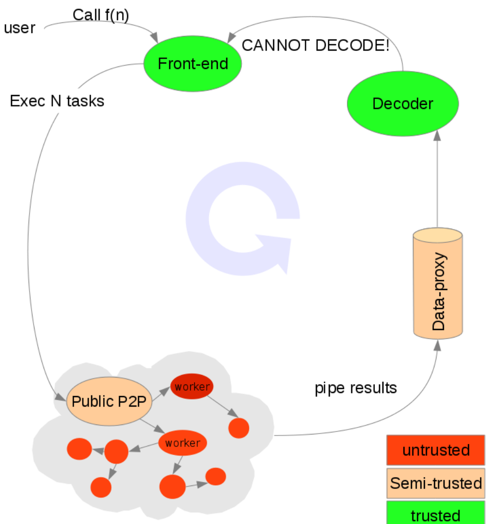
3 types of resources:

- Trusted
  - 100% secure
- Untrusted
  - no security measures (public machines)
- Semi-Trusted
  - accessible by untrusted machines but with security measures
  - firewall, chroot, input checking...etc

# Basic scheme

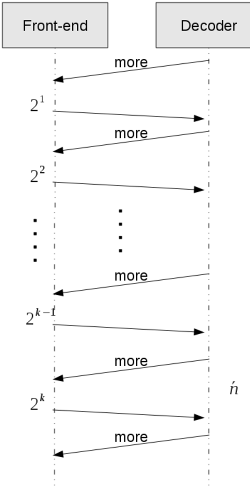


# Basic scheme



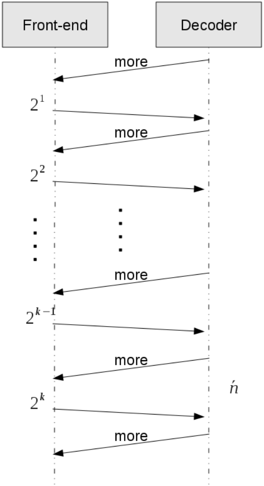
# Basic scheme

- we don't know the number of tasks!
- assuming the error rate is not fixed
- Trade-off: iterations vs #forks



# Basic scheme

- we don't know the number of tasks!
- assuming the error rate is not fixed
- Trade-off: iterations vs #forks

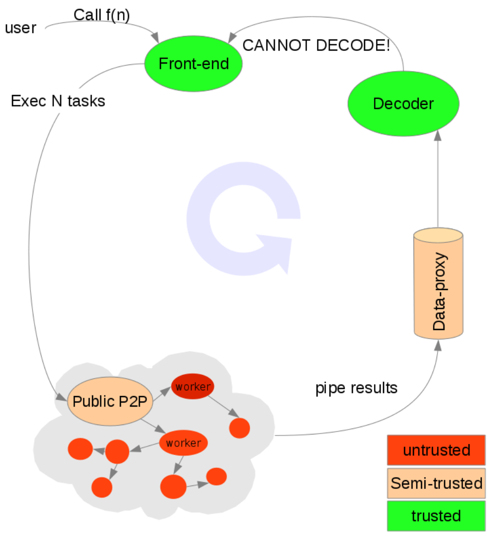



$\acute{n}$

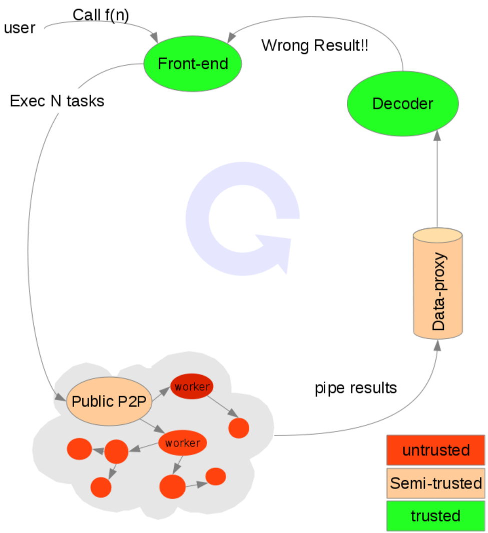
increasing iteratively the number of tasks:

- quadratic growth: $2^i$
- $\rho^i$ with $1 < \rho < 2$
- $\rho$-amortized: $\rho^{f(i)}$ with $f$ sub-linear
  - $f(i) = i^\alpha$ with $0 < \alpha < 1$
  - $f(i) = \frac{i}{\log i}$, ....

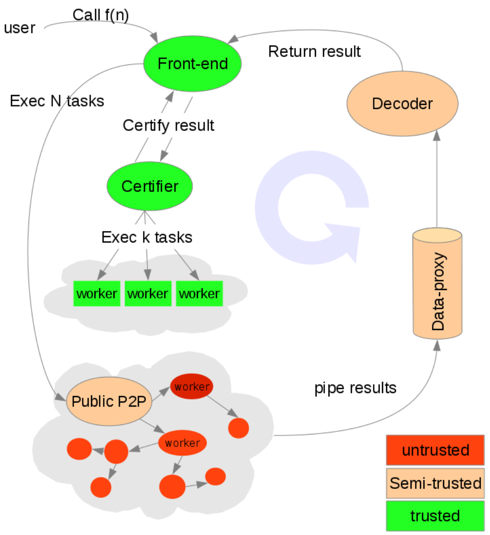
# Online Post certification scheme



# Online Post certification scheme

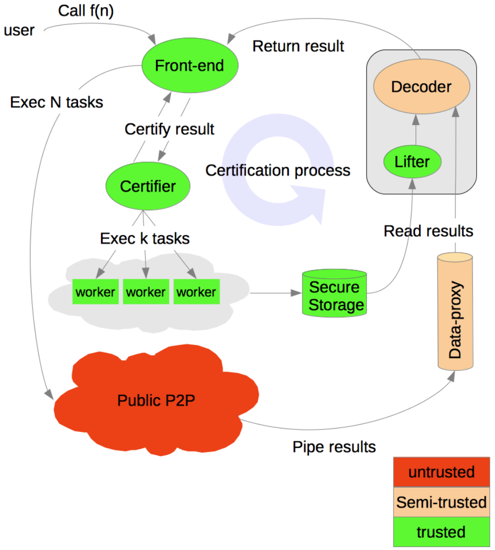


# Online Post certification scheme



# Certification process

Read the decoder result $R'$
pick up a random prime $p_i$
compute $r = f$ over $\mathbb{Z}_{p_i}$
reduce $r' = R' \mod p_i$
**if** $r! = r'$ **then**
  Return ERROR.
**else**
  Return SUCCESS with a given probability
**end if**


Call f(n)
user
Front-end
Return result
Decoder
Exec N tasks
Certify result
Lifter
Certifier
Certification process
Read results
Exec k tasks
worker
worker
worker
Secure Storage
Data-proxy
Public P2P
Pipe results
untrusted
Semi-trusted
trusted

# Outline





Majid Khonji and Clément Pernet

Inroduction
Distributed and Cloud computing
Fault tolerance
Parallel Exact linear algebra
Chinese Remainder and Error correcting code
Principle
Correction
Overview of the computation scheme
Further improvements
Ordering, sign
Early termination
Updated computation scheme
Application to computing Determinants
Softwares
Live demo
Perspectives

# Removing the ordering of the moduli

worst case scenario :

every errors having a bigger contribution *at the end*,

Mandelbaum algorithm requires to sort the moduli $p_i$:

- to ensures accurately the $t = (n - k)/2$ correction rate
- to compute the optimal bound $C_{\max} = \prod_{i=n-k+1}^{n} p_i$ for the search of convergents.

# Removing the ordering of the moduli

worst case scenario :

every errors having a bigger contribution *at the end*,

Mandelbaum algorithm requires to sort the moduli $p_i$:

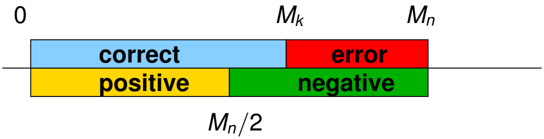
- to ensures accurately the $t = (n - k)/2$ correction rate
- to compute the optimal bound $C_{\max} = \prod_{i=n-k+1}^{n} p_i$ for the search of convergents.

Relaxation:

- no sort, only find $p_{\min}$ and $p_{\max}$,
- more suited for a future *on-the-fly* model
- only an approximation of the distance (off by 1 in the worst cases)
- less accurate $C_{\max}$,

# Dealing with negative numbers

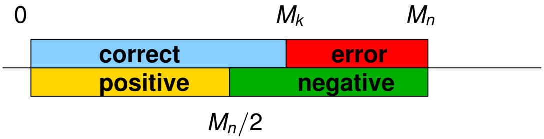
$$x \in C \text{ iff } 0 \leq x < M_k$$




⇒Mandelbaum algorithm only works with positive numbers.

# Dealing with negative numbers

$$x \in C \text{ iff } 0 \leq x < M_k$$



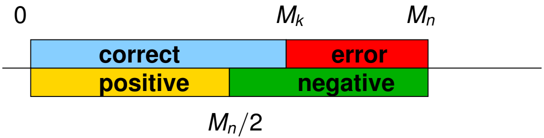

$\Rightarrow$Mandelbaum algorithm only works with positive numbers.

$(1 \mod 5, 1 \mod 7, 10 \mod 11)$ could be

- $-34$ with 0 errors
- 1 with 1 error

# Dealing with negative numbers

$$x \in C \text{ iff } 0 \leq x < M_k$$




⇒Mandelbaum algorithm only works with positive numbers.

$(1 \mod 5, 1 \mod 7, 10 \mod 11)$ could be

- $-34$ with 0 errors
- 1 with 1 error

Work-around solution:

- reconstruct both

$$x = CRT(x_1, \ldots, x_n)$$
$$y = CRT(-x_1, \ldots, -x_n)$$

- discard the bad one with the certifier

# Early termination

Chinese Remainder Algorithm requires a bound.
For Determinant ⇒Hadamard:

$$|\det(A)| \leq n^n (\max |a_{i,j}|)^{n/2} \leq \prod_i p_i$$

# Early termination

Chinese Remainder Algorithm requires a bound.
For Determinant ⇒Hadamard:

$$|\det(A)| \leq n^n (\max |a_{i,j}|)^{n/2} \leq \prod_i p_i$$

Could be **really** pessimistic with

- sparse/structured matrices,
- matrices with special properties,...

# Early termination

Chinese Remainder Algorithm requires a bound.
For Determinant ⇒Hadamard:

$$|\det(A)| \leq n^n (\max |a_{i,j}|)^{n/2} \leq \prod_i p_i$$

Could be **really** pessimistic with

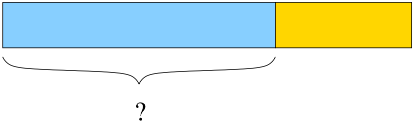
- sparse/structured matrices,
- matrices with special properties,...

⇒**discover the bound during the computation**

- for each new modular result, reconstruct the result
- if it stabilizes $k$ times ⇒correct result with probability $1 - 1/p^k$

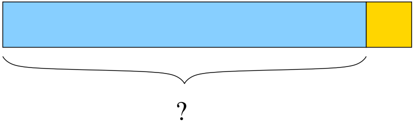
# Early termination

**Problem:** how to detect errors without a bound?



# Early termination

Problem: how to detect errors without a bound?

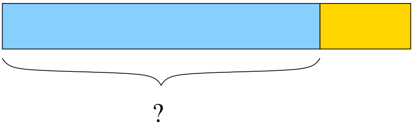



Multiple solution candidates:

- Solution $x_1$ with 1 error corrected

# Early termination

**Problem:** how to detect errors without a bound?

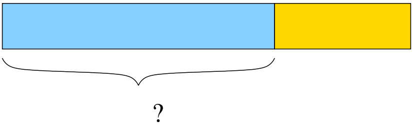



Multiple solution candidates:

- Solution $x_1$ with 1 error corrected
- Solution $x_2$ with 3 error corrected

# Early termination

Problem: how to detect errors without a bound?

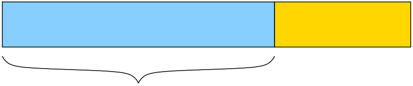



Multiple solution candidates:

- Solution $x_1$ with 1 error corrected
- Solution $x_2$ with 3 error corrected
- Solution $x_3$,with 2 errors corrected

# Early termination

Problem: how to detect errors without a bound?

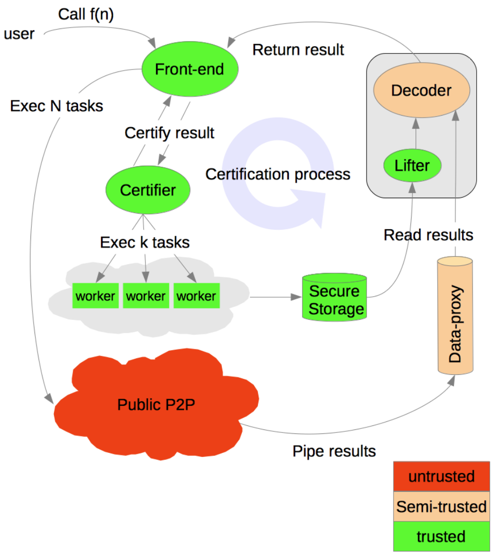



Multiple solution candidates:

- Solution $x_1$ with 1 error corrected
- Solution $x_2$ with 3 error corrected
- Solution $x_3$,with 2 errors corrected

The certifier discriminates the candidates

# Computation scheme



# Multiple decoding

**sign:** 2 candidates

**early term.:** multiple candidates, for each possible bound

Decoder returns a $l$ of candidates
**while** $|l| > 1$ **do**
  pick a random prime $p_i$
  Certifier compute $f_i = f \mod p_i$
  sieve un-compatible elements in $l$
  **if** $|l| = 1$ **then**
    return l.front()
  **end if**
**end while**

# Outline

# Parallelism : `Kaapi`

**Kernel for Adaptive Asynchronous Parallel Interface**

Virtualization of the parallel platform

- SMP, distributed architectures
- virtualizes the processors
  $\Rightarrow$programmer only focus on a high level description of the parallelism
- Scheduling:
  - static optimization from the data flow graph
  - dynamic using work stealing at fine grain granularity

# Parallelism : Kaapi
## Example of code

```
struct Fibonacci {
  void operator()( int n,              int& result )
    {
        if (n < 2) result =      n  ;
        else {
                       int  subresult1;
                       int  subresult2;
                     Fibonacci ()(n-1, subresult1);
                     Fibonacci ()(n-2, subresult2);
                     Sum ()(result, subresult1, subresult2);
        }
    }
};

struct Sum {
  void operator()(             int& result,
                          int  sr1,
                          int  sr2 )
    { result =      sr1        + sr2         ; }
}
```

31

# Parallelism : Kaapi

## Example of code

```
struct Fibonacci {
  void operator()( int n, a1::Shared_w<int> result )
    {
        if (n < 2) result.write( n );
        else {
            a1::Shared<int> subresult1;
            a1::Shared<int> subresult2;
            a1::Fork<Fibonacci>()(n-1, subresult1);
            a1::Fork<Fibonacci>()(n-2, subresult2);
            a1::Fork<Sum>()(result, subresult1, subresult2);
        }
    }
};

struct Sum {
  void operator()( a1::Shared_w<int> result,
                   a1::Shared_w<int> sr1,
                   a1::Shared_w<int> sr2 )
    { result.write( sr1.read() + sr2.read() ); }
}
```

32

# Computations

## Linear Algebra tasks: `LinBox`

- Fast MatMul (BLAS, delayed modular reductions, Strassen,...)
- Efficient, in-place, block Gaussian elimination

# Computations

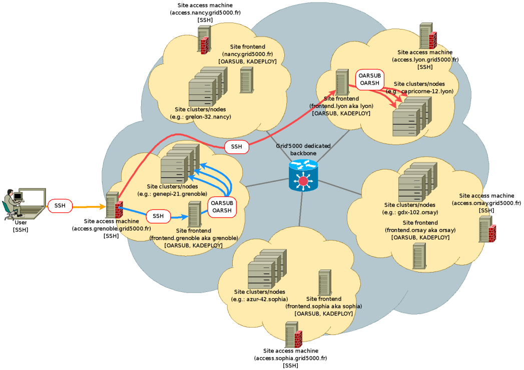
## Linear Algebra tasks: `LinBox`

- Fast MatMul (BLAS, delayed modular reductions, Strassen,...)
- Efficient, in-place, block Gaussian elimination

## Decoder: `Sage`

- ease of use for experimenting different prototypes
- good efficiency
- but BIG executable $\Rightarrow$hard to distribute

# The Grid 5000 architecture



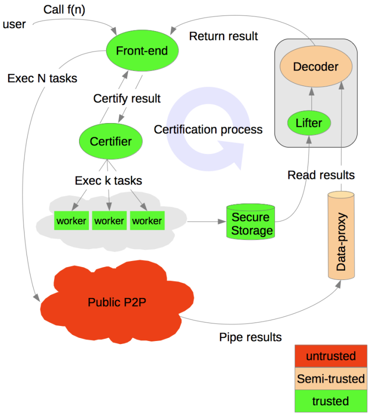

- Country-wide grid of 9 sites (Nancy, Lyon, Grenoble, Toulouse, Lille, Bordeaux, Orsay, Rennes, Sophia)
- Each site has 2-3 clusters of 60-600 computers
- heterogeneous architecture
  $\Rightarrow$ 3202 processors, 5714 cores

# Live demo

# Outline

# Perspectives

## A fully on-the-fly computation model:




- Need a fast CRT working *on the fly*
- Refine the amortized technique

# Sage ?

Revive the Dsage efforts?

distributed computations: Kaapi optional package?

- resource oblivious
- task parallelism, work-stealing, ABFT
- perspectives for GPU/multi-CPU support

P2P, volunteer computing

*Spare your cycles and contribute solving BSD conjecture!*