Distributed hybrid Gröbner bases computation

Heinz Kredel
University of Mannheim

ECDS at CISIS 2010, Krakow
Overview

• Introduction to JAS
• Gröbner bases
  • sequential and parallel algorithm
  • problems with parallel computation
• Distributed and distributed hybrid algorithm
  • execution middle-ware
  • data structure middle-ware
• Evaluation
  • termination, selection strategies, hardware
• Conclusions and future work
Java Algebra System (JAS)

- object oriented design of a computer algebra system
  = software collection for symbolic (non-numeric) computations
- type safe through Java generic types
- thread safe, ready for multi-core CPUs
- use dynamic memory system with GC
- 64-bit ready
- jython (Java Python) interactive scripting front end
Implementation overview

- 250+ classes and interfaces
- plus ~120 JUnit test classes, 3800+ assertion tests
- uses JDK 1.6 with generic types
  - Javadoc API documentation
  - logging with Apache Log4j
  - build tool is Apache Ant
  - revision control with Subversion
  - public git repository
- jython (Java Python) scripts
  - support for Sage like polynomial expressions
- open source, license is GPL or LGPL
Example: Legendre polynomials

\[
P[0] = 1; \quad P[1] = x; \\
P[i] = \frac{1}{i} ( (2i-1) \times x \times P[i-1] - (i-1) \times P[i-2] )
\]

```java
BigRational fac = new BigRational();
String[] var = new String[]{ "x" };
GenPolynomialRing<BigRational> ring
    = new GenPolynomialRing<BigRational>(fac,1,var);
List<GenPolynomial<BigRational>> P
    = new ArrayList<GenPolynomial<BigRational>>(n);
GenPolynomial<BigRational> t, one, x, xc, xn; BigRational n21, nn;

one = ring.getONE(); x = ring.univariate(0);
P.add( one ); P.add( x );
for ( int i = 2; i < n; i++ ) {
    n21 = new BigRational( 2*i-1 );  xc = x.multiply( n21 );
    t = xc.multiply( P.get(i-1) );
    nn = new BigRational( i-1 ); xc = P.get(i-2).multiply( nn );
    t = t.subtract( xc );  nn = new BigRational(1,i);
    t = t.multiply( nn ); P.add( t );
}
int i = 0;
for ( GenPolynomial<BigRational> p : P ) {
    System.out.println("P["+(i++)+"] = " + p);
}
```
Overview

- Introduction to JAS
- Gröbner bases
  - sequential and parallel algorithm
  - problems with parallel computation
- Distributed and distributed hybrid algorithm
  - execution middle-ware
  - data structure middle-ware
- Evaluation
  - termination, selection strategies, hardware
- Conclusions and future work
Gröbner bases

- canonical bases in polynomial rings $R = C[x_1, \ldots, x_n]$
- like Gauss elimination in linear algebra
- like Euclidean algorithm for univariate polynomials
- with a Gröbner base many problems can be solved
  - solution of non-linear systems of equations
  - existence of solutions
  - solution of parametric equations
- slower than multivariate Newton iteration in numerics
- but in computer algebra no round-off errors
- so guarantied correct results
Buchberger algorithm

algorithm: \text{GB}( F )
input: F a list of polynomials in \( \mathbb{R}[x_1, \ldots, x_n] \)
output: G a Gröbner Base of ideal(F)

\[ \begin{align*}
G &= F; \\
B &= \{ (f,g) \mid f, g \in G, f \neq g \}; \\
\text{while} \ ( B \neq \{ \} ) \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { } \ { }
Problems with the GB algorithm

- requires exponential space (in the number of variables)
- even for arbitrary many processors no polynomial time algorithm will exist
- highly data depended
  - number of pairs unknown (size of B)
  - size of polynomials s and h unknown
    - size of coefficients
    - degrees, number of terms
- management of B is sequential
- strategy for the selection of pairs from B
  - depends moreover on speed of reducers
GröbnerBase
+ isGB(F : List<GenPolynomial>) : boolean
+ GB(F : List<GenPolynomial>) : List<GenPolynomial>
+ extGB(F : List<GenPolynomial>) : ExtendedGB
+ minimalGB(G : List<GenPolynomial>) : List<GenPolynomial>

Reduction
+ normalform(F : List<GenPolynomial>, p : GenPolynomial) : GenPolynomial

GröbnerBaseAbstract
+ GrobnerBaseAbstract(red : Reduction)
+ isGB(F : List<GenPolynomial>) : boolean
+ isGB(modv : int, F : List<GenPolynomial>) : boolean
+ GB(F : List<GenPolynomial>) : List<GenPolynomial>
+ GB(modv : int, F : List<GenPolynomial>) : List<GenPolynomial>
+ extGB(F : List<GenPolynomial>) : ExtendedGB
+ extGB(modv : int, F : List<GenPolynomial>) : ExtendedGB
+ minimalGB(G : List<GenPolynomial>) : List<GenPolynomial>

GröbnerBaseSeq
+ GrobnerBaseSeq(red : Reduction)
+ GB(modv : int, F : List<GenPolynomial>) : List<GenPolynomial>

GröbnerBaseParallel
+ GrobnerBaseParallel(threads : int, red : Reduction)
+ GB(modv : int, F : List<GenPolynomial>) : List<GenPolynomial>

GröbnerBaseDistributed
+ GrobnerBaseDistributed(threads : int, red : Reduction, port : int)
+ GB(modv : int, F : List<GenPolynomial>) : List<GenPolynomial>

GröbnerBaseDistributedHybrid
+ GrobnerBaseDistributedHybrid(threads : int, tpernode : int, red : Reduction, port : int)
+ GB(modv : int, F : List<GenPolynomial>)
Overview

- Introduction to JAS
- Gröbner bases
  - sequential and parallel algorithm
  - problems with parallel computation
- Distributed and distributed hybrid algorithm
  - execution middle-ware
  - data structure middle-ware
- Evaluation
  - termination, selection strategies, hardware
- Conclusions and future work
bwGRiD cluster architecture

- 8-core CPU nodes @ 2.83 GHz, 16GB, 140 nodes
- shared Lustre home directories
- 10Gbit InfiniBand and 1Gbit Ethernet interconnects
- managed by PBS batch system with Maui scheduler
- running Java 64bit server VM 1.6 with 4+GB memory
- start Java VMs with daemons on allocated nodes
- communication via TCP/IP interface over InfiniBand
- no Java high performance interface to InfiniBand
- alternative Java via MPI not studied
- other middle-ware ProActive or GridGain not studied
Distributed hybrid GB algorithm

- main method $\text{GB}()$
- distribute list $G$ via distributed hash table (DHT)
- start $\text{HybridReducerServer}$ threads for each node
  - together with a $\text{HybridReducerReceiver}$ thread
- $\text{clientPart}()$ starts multiple $\text{HybridReducerClient}$s threads
- establish one control network connection per node
- select pair and send to distributed client
  - send index of polynomial in $G$
- clients perform S-polynomial and normalform computation send result back to master
- master eventually inserts new pairs to $B$ and adds polynomial to $G$ in DHT
Thread to node mapping

**master node**
- critical pairs
- idle count
- reducer server
- reducer receiver
- DHT server
- DHT

**multi-CPU nodes**
- reducer server
- reducer receiver
- DHT

one connection per node
Middleware overview

GBDist

Distributed ThreadPool

Reducer Server

GB()

DHT Client

DHT Server

ExecutableServer

Distributed Thread

Reducer Client

clientPart()

DHT Client

master node

InfiniBand

a client node
Execution middle-ware (nodes)

- on compute nodes do basic bootstrapping
  - start daemon class `ExecutableServer`
  - listens on connections (no security constrains)
  - start thread with `Executor` for each connection
  - receives (serialized) objects with `RemoteExecutable` interface
  - execute the `run()` method
  - communication and further logic is implemented in the `run()` method
- multiple processes as threads in one JVM

same as for distributed algorithm
Execution middle-ware (master)

- **start** `DistThreadPool` **similar to** `ThreadPool`
- starts threads for each compute node
- list of compute nodes taken from PBS
- starts connections to all nodes with `ExecutableChannel`
- can start multiple tasks on nodes to use multiple CPU cores via `open(n)` method
- **method** `addJob()` on master
- send a job to a remote node and wait until termination (RMI like)

same as for distributed algorithm
Execution middle-ware usage
mostly same as for distributed algorithm

- Gröbner base master `GBDistHybrid`
- initialize `DistThreadPool` with PBS node list
- initialize `GroebnerBaseDistributedHybrid`
- `execute()` method of `GBDistHybrid`
  - add remote computation classes as jobs
  - `execute clientPart()` method in jobs
    - is `HybridReducerClient` above
  - calls main `GB()` method
    - `start HybridReducerServer` above
    - which then `starts HybridReducerReceiver`
Communication middle-ware

- one (TCP/IP) connection per compute node
- request and result messages can overlap
- solved with tagged message channel
  - message is tagged with a label, so receive() can select messages with specific tags
- implemented in class TaggedSocketChannel
- methods with tag parameter
  - send(tag, object) and receive(tag)
- implemented with blocking queues for each tag and a separate receiving thread
- alternative: java.nio.channels.Selector
Data structure middle-ware

- sending of polynomials involves
  - serialization and de-serialization time
  - and communication time
- avoid sending via a distributed data structure
- implemented as distributed list
- runs independently of main GB master
- **setup in** `GroebnerBaseDistributedHybrid constructor` and `clientPart()` method
- then only indexes of polynomials need to be communicated
Distributed polynomial list

- distributed list implemented as distributed hash table (DHT)
- key is list index
- implemented with generic types
- class `DistHashTable` extends `java.util.AbstractMap`
- methods `clear()`, `get()` and `put()` as in `HashMap`
- method `getWait(key)` waits until a value for a key has arrived
- method `putWait(key, value)` waits until value has arrived at the master and is received back
- no guaranty that value is received on all nodes
DHT implementation (1)

• implemented as central control DHT
• client part on node uses TreeMap as store
• client DistributedHashTable connects to master
• master class DistributedHashTableServer
• put() methods send key-value pair to a master
• master then broadcasts key-value pair to all nodes
• get() method takes value from local TreeMap
• in future implement DHT with decentralized control

improved version
DHT implementation (2)

- in master process de-serialization of polynomials is now avoided
- broadcast to clients in master now use serialized polynomials in marshaled objects
- master is co-located to master of GB computation on same compute node
- this doubles memory requirements on master node
- this increases the CPU load on the master
  - limits scaling of master for more nodes

improved version
Marshalled objects

- reduce serialization overhead in DHT for polynomials
- use class `MarshalledObject` from `java.rmi`
- polynomials on DHT master are no more de-serialized and re-serialized
- serialization and de-serialization takes place only upon entry and exit in client side DHT
- timing samples from distributed and hybrid GB
  - sum of encoding and decoding
  - plus sum of marshalled object encoding and decoding

<table>
<thead>
<tr>
<th>example</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>plain</td>
<td>2461</td>
<td>2364</td>
<td>1289</td>
<td>1100</td>
</tr>
<tr>
<td>marshall</td>
<td>487</td>
<td>765</td>
<td>394</td>
<td>594</td>
</tr>
</tbody>
</table>
Overview

- Introduction to JAS
- Gröbner bases
  - sequential and parallel algorithm
  - problems with parallel computation
- Distributed and distributed hybrid algorithm
  - execution middle-ware
  - data structure middle-ware
- Evaluation
  - termination, selection strategies, hardware
- Conclusions and future work
Termination (1)

- single thread can check if B is empty
- tests in case of multiple threads
  - B is empty
  - and all threads are idle
- distributed hybrid termination
  - idle client requests critical pair
  - thread on master waits for such requests, then
    - if B is empty and all threads are idle then terminate
    - if B is not empty then take pair and send to reducer client
    - if B is empty and threads are working, then sleep and recheck on wake-up
- thread on master responsible for multiple node threads
Termination (2)

- Critical-pairs
- Idle-count
- Server
- Receiver
- Client

1: Request pair
2: Decrement
3: Retrieve pair
4: Send result
5: Record result
6: Increment
7: Request next pair
Termination (3)

- multiple requests over the same connection
- **uses** TaggedSocketChannel
- send critical pair: receiving thread may not be the same as requesting thread
- pair handling thread may be blocked for requests
- so helper thread `HybridReducerReceiver` for result polynomials is required
  - record the result in the pair-list data structure
  - update idle threads count
  - send back acknowledgment
  - need to identify exact receiving thread: message tag
Termination (4)

- processing sequence in a master thread
  - receive reduction request
  - update idle threads count
  - retrieve a critical pair and update the pair-list
  - send pair-index to client
- acknowledgment ensures that the reduction request does not overlap with the other steps
- acknowledgment reduces parallelism, but required for book-keeping
Termination (5)

- processing sequence of client reducer thread
  - send pair request to master
  - receive pair index
  - process pair
    - retrieve polynomials from DTH via index
    - compute S-polynomial and a normal form
  - send result polynomial to master receiver
  - wait for acknowledgment from master
<table>
<thead>
<tr>
<th>1</th>
<th>(1,0,0,0,0,0,1,0):long</th>
<th>1</th>
<th>(1,0,0,0,0,0,1,0):long</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(1,0,0,0,0,1,0,0):long</td>
<td>2</td>
<td>(1,0,0,0,0,1,0,0):long</td>
</tr>
<tr>
<td>3</td>
<td>(1,0,0,0,1,0,0,0):long</td>
<td>3</td>
<td>(1,0,0,0,1,0,0,0):long</td>
</tr>
<tr>
<td>4</td>
<td>(1,0,0,1,0,0,0,0):long</td>
<td>4</td>
<td>(1,0,0,1,0,0,0,0):long</td>
</tr>
<tr>
<td>5</td>
<td>(1,0,1,0,0,0,0,0):long</td>
<td>5</td>
<td>(1,0,1,0,0,0,0,0):long</td>
</tr>
<tr>
<td>6</td>
<td>(1,1,0,0,0,0,0,0):long</td>
<td>6</td>
<td>(1,1,0,0,0,0,0,0):long</td>
</tr>
<tr>
<td>7</td>
<td>(2,0,0,0,0,0,0,0):long</td>
<td>7</td>
<td>(2,0,0,0,0,0,0,0):long</td>
</tr>
<tr>
<td>8</td>
<td>(0,1,0,0,0,1,1,0):long</td>
<td>8</td>
<td>(0,1,0,0,0,1,1,0):long</td>
</tr>
<tr>
<td>9</td>
<td>(0,1,0,1,0,0,0,1):long</td>
<td>9</td>
<td>(0,1,0,1,0,0,0,1):long</td>
</tr>
<tr>
<td>10</td>
<td>(0,1,0,1,0,1,0,0):long</td>
<td>10</td>
<td>(0,1,0,1,0,1,0,0):long</td>
</tr>
<tr>
<td>11</td>
<td>(0,1,0,1,1,0,0,0):long</td>
<td>11</td>
<td>(0,2,0,0,0,0,0,0):long</td>
</tr>
<tr>
<td>12</td>
<td>(0,1,0,1,0,0,0,0):long</td>
<td>12</td>
<td>(0,1,0,1,0,0,0,0):long</td>
</tr>
<tr>
<td>13</td>
<td>(0,1,0,1,0,1,0,0):long</td>
<td>13</td>
<td>(0,1,0,1,1,0,0,0):long</td>
</tr>
<tr>
<td>14</td>
<td>(0,2,0,0,0,0,0,0):long</td>
<td>14</td>
<td>(0,1,0,1,0,0,0,0):long</td>
</tr>
<tr>
<td>15</td>
<td>(0,1,0,1,1,0,0,0):long</td>
<td>15</td>
<td>(0,1,1,0,0,0,0,1):long</td>
</tr>
<tr>
<td>16</td>
<td>(0,1,1,0,0,1,0,0):long</td>
<td>16</td>
<td>(0,1,1,0,0,1,0,0):long</td>
</tr>
<tr>
<td>17</td>
<td>(0,1,1,0,1,0,0,0):long</td>
<td>17</td>
<td>(0,1,1,0,1,0,0,0):long</td>
</tr>
<tr>
<td>18</td>
<td>(0,1,1,1,0,0,0,0):long</td>
<td>18</td>
<td>(0,1,1,1,0,0,0,0):long</td>
</tr>
<tr>
<td>19</td>
<td>(0,1,0,0,0,0,1,1):long</td>
<td>19</td>
<td>(0,2,0,0,0,0,0,1):long</td>
</tr>
<tr>
<td>20</td>
<td>(0,1,0,0,0,1,0,1):long</td>
<td>20</td>
<td>(0,2,0,0,0,0,1,0):long</td>
</tr>
<tr>
<td>21</td>
<td>(0,1,0,0,1,0,0,1):long</td>
<td>21</td>
<td>(0,2,0,0,0,1,0,0):long</td>
</tr>
<tr>
<td>22</td>
<td>(0,0,1,1,0,1,0,1):long</td>
<td>22</td>
<td>(0,2,0,0,0,1,0,0):long</td>
</tr>
<tr>
<td>23</td>
<td>(0,1,0,1,0,0,0,0):long</td>
<td>23</td>
<td>(0,2,0,0,1,0,0,0):long</td>
</tr>
<tr>
<td>24</td>
<td>(0,1,1,0,0,0,0,1):long</td>
<td>24</td>
<td>(0,2,0,0,1,0,0,0):long</td>
</tr>
<tr>
<td>25</td>
<td>(0,2,0,0,0,0,0,1):long</td>
<td>25</td>
<td>(0,2,0,0,1,0,0,0):long</td>
</tr>
<tr>
<td>26</td>
<td>(0,2,0,0,0,0,0,1):long</td>
<td>26</td>
<td>(0,1,0,0,0,0,1,1):long</td>
</tr>
<tr>
<td>27</td>
<td>(0,2,0,0,0,1,0,0):long</td>
<td>27</td>
<td>(0,1,0,0,0,1,0,1):long</td>
</tr>
<tr>
<td>28</td>
<td>(0,2,0,0,0,1,0,0):long</td>
<td>28</td>
<td>(0,1,0,0,1,0,0,1):long</td>
</tr>
<tr>
<td>29</td>
<td>(0,2,0,0,1,0,0,0):long</td>
<td>29</td>
<td>(0,1,0,1,0,0,0,0):long</td>
</tr>
<tr>
<td>30</td>
<td>(0,2,0,0,1,0,0,0):long</td>
<td>30</td>
<td>(0,0,1,1,0,1,0,0):long</td>
</tr>
<tr>
<td>31</td>
<td>(0,2,0,0,1,0,0,0):long</td>
<td>31</td>
<td>(0,1,1,0,0,0,0,1):long</td>
</tr>
<tr>
<td>32</td>
<td>(0,2,0,0,1,0,0,0):long</td>
<td>32</td>
<td>(0,2,0,0,0,0,0,1):long</td>
</tr>
<tr>
<td>33</td>
<td>(0,2,0,1,0,0,0,0):long</td>
<td>33</td>
<td>(0,2,0,0,0,0,0,1):long</td>
</tr>
<tr>
<td>34</td>
<td>(0,0,2,0,0,0,0,1):long</td>
<td>34</td>
<td>(0,2,0,1,0,0,0,0):long</td>
</tr>
<tr>
<td>35</td>
<td>(0,2,0,1,0,0,0,0):long</td>
<td>35</td>
<td>(0,2,1,0,0,0,0,0):long</td>
</tr>
<tr>
<td>36</td>
<td>(0,2,1,0,0,0,0,0):long</td>
<td>36</td>
<td>(0,2,1,0,0,0,0,0):long</td>
</tr>
<tr>
<td>37</td>
<td>(0,2,1,0,0,0,0,0):long</td>
<td>37</td>
<td>(1,0,0,0,0,1,1,0):long</td>
</tr>
<tr>
<td>38</td>
<td>(1,0,0,0,0,1,1,0):long</td>
<td>38</td>
<td>(1,0,0,0,1,0,1,0):long</td>
</tr>
<tr>
<td></td>
<td>Long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(0,1,0,0,0,0,1,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(0,1,1,0,0,0,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(0,1,0,1,0,0,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(0,2,0,0,0,0,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>(0,1,0,0,1,0,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>(0,1,0,0,1,0,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>(0,0,1,0,1,1,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>(0,0,0,0,1,0,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>(0,0,1,0,1,0,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>(0,1,0,0,0,0,0,1):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>(0,0,1,0,0,2,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>(0,0,1,0,1,1,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>(0,0,1,0,0,0,2,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>(0,0,1,0,0,1,0,1):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>(0,0,1,0,1,0,0,1):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>(0,0,1,1,0,1,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>(0,0,1,0,1,0,0,1):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>(0,0,2,0,0,0,1,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>(0,0,2,0,0,0,0,1):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>(0,0,2,0,0,0,0,2):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>(0,0,1,0,2,0,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>(0,0,1,0,0,0,1,1):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>(0,0,1,0,0,0,1,1):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>(0,0,2,0,1,0,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>(0,0,2,0,0,0,1,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>(0,0,1,1,1,0,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>(0,0,1,2,0,0,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>(0,0,1,0,0,0,0,2):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>(0,0,2,0,1,1,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>(0,0,2,1,0,0,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>(0,0,3,0,0,0,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>(0,0,2,0,1,0,1):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>(0,0,2,0,0,2,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>(0,0,2,0,0,2,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>(0,0,1,1,2,0,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>(0,0,1,0,2,0,1,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>(0,0,0,1,0,0,1,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>(0,0,0,1,1,1,0,1):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>(0,0,0,1,1,1,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>(0,0,0,2,1,1,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>(0,0,1,1,0,2,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>(0,0,0,2,1,1,0,0):long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>(0,0,0,3,1,1,0,0):long</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Selection strategies (1)

- best to use the same order of polynomials and pairs as in sequential algorithm
- selection algorithm is sequential
  - so optimizations reduce parallelism
- Attardi & Traverso: 'strategy-accurate' algorithm
  - rest reduction sequential
  - only top-reduction in parallel
Selection strategies (2)

- Amrhein & Gloor & Küchlin:
  - work parallel: n reductions in parallel
  - search parallel: select best from k results
- Kredel:
  - n reductions in parallel, select first finished
  - select result in same sequence as reduction is started, not the first finished
Hardware

- InfiniBand 10Gbit node to node
- 1 Gbit Ethernet shared between 14 nodes
- use TCP/IP stack on InfiniBand
- bypass TCP/IP stack eventually in JDK 1.7
  - JAS doesn't compile on JDK 1.7 due to compiler bug
Conclusions

- first version of a distributed hybrid GB algorithm
- runs on a HPC cluster in PBS environment
- shared memory parallel version scales up to 8 CPUs
- runtime of distributed version is comparable to parallel version, speed-up of ~4
- runtime of distributed hybrid is comparable to distributed version, speed-up of ~4
- reduced communication between nodes, shared channels
- serialization overhead reduced with marshaled objects
- less memory required on nodes comp. dist. version
- new package is now type-safe with generic types
Future work

- profile and study run-time behavior in detail
- investigate other grid middleware
- improve integration into the grid environment
- study other result selection strategies
- compute sequential Gröbner bases with respect to different term orders in parallel
- test with JDK 1.7
- test other examples
Thank you

• Questions or Comments?
• http://krum.rz.uni-mannheim.de/jas

• Thanks to
  • Raphael Jolly
  • Thomas Becker
  • Hans-Günther Kruse
  • bwGRiD for providing computing time
  • the referees
  • and other colleagues