Parallel computations of Gröbner bases in the Weyl algebra

Something to run on a machine with 128 cores

Anton Leykin

Institute for Mathematics and its Applications, Minneapolis

MSRI, Berkeley, 2007

Definition (n-th Weyl algebra over field K of characteristic 0)

$$D = A_n(K) = K\langle x, \partial\rangle = K\langle x_1, \partial_1, \ldots, x_n, \partial_n\rangle,$$

where $[\partial_i, x_i] = \partial_i x_i - x_i \partial_i = 1$ and all other pairs commute.

Multiplication in Weyl algebra: Leibnitz rule

$A_n = K\langle x_1, \ldots, x_n, \partial_1, \ldots, \partial_n\rangle$ then for $P, Q \in A_n$

$$PQ = \sum_{\alpha \in \mathbb{Z}^n_{\geqslant 0}} \frac{1}{\alpha!} \mathrm{Diff}(P, \partial^\alpha) * \mathrm{Diff}(Q, x^\alpha),$$

where Diff is a formal partial derivative (as if P, Q are polynomials) and $*$ is the polynomial multiplication.

Weyl algebra in computer algebra systems

kan/sm1, risa/asir (Takayama, Noro); Macaulay 2 (Grayson, Stillman), D-modules for M2 (A.L., Tsai); Singular/Plural (Levandovskyy); CoCoA (group in Genova, Italy).

Definition (n-th Weyl algebra over field K of characteristic 0)

$$D = A_n(K) = K\langle x, \partial \rangle = K\langle x_1, \partial_1, \dots, x_n, \partial_n \rangle,$$

where $[\partial_i, x_i] = \partial_i x_i - x_i \partial_i = 1$ and all other pairs commute.

Multiplication in Weyl algebra: Leibnitz rule

$A_n = K\langle x_1, \dots, x_n, \partial_1, \dots, \partial_n \rangle$ then for $P, Q \in A_n$

$$PQ = \sum_{\alpha \in \mathbb{Z}^n_{\geqslant 0}} \frac{1}{\alpha!} \mathrm{Diff}(P, \partial^\alpha) * \mathrm{Diff}(Q, x^\alpha),$$

where Diff is a formal partial derivative (as if P, Q are polynomials) and $*$ is the polynomial multiplication.

Weyl algebra in computer algebra systems

kan/sm1, risa/asir (Takayama, Noro); Macaulay 2 (Grayson, Stillman), D-modules for M2 (A.L., Tsai); Singular/Plural (Levandovskyy); CoCoA (group in Genova, Italy).

Definition (n-th Weyl algebra over field K of characteristic 0)

$$D = A_n(K) = K\langle x, \partial \rangle = K\langle x_1, \partial_1, \dots, x_n, \partial_n \rangle,$$

where $[\partial_i, x_i] = \partial_i x_i - x_i \partial_i = 1$ and all other pairs commute.

Multiplication in Weyl algebra: Leibnitz rule

$A_n = K\langle x_1, \dots, x_n, \partial_1, \dots, \partial_n \rangle$ then for $P, Q \in A_n$

$$PQ = \sum_{\alpha \in \mathbb{Z}^n_{\geqslant 0}} \frac{1}{\alpha!} \mathrm{Diff}(P, \partial^\alpha) * \mathrm{Diff}(Q, x^\alpha),$$

where Diff is a formal partial derivative (as if P, Q are polynomials) and $*$ is the polynomial multiplication.

Weyl algebra in computer algebra systems

kan/sm1, risa/asir (Takayama, Noro); Macaulay 2 (Grayson, Stillman), D-modules for M2 (A.L., Tsai); Singular/Plural (Levandovskyy); CoCoA (group in Genova, Italy).

Let R be a Gröbner-friendly algebra (think: $R = K[x_1, \dots, x_n]$).

Definition

Given a fixed *admissible* monomial ordering, a polynomial $f \in R$ has

- initial monomial $\mathrm{lm}(f)$;
- initial coefficient $\mathrm{lc}(f)$;
- initial term $\mathrm{lt}(f) = \mathrm{lc}(f)\,\mathrm{lm}(f)$.

Algorithm $REDUCE(f, B)$

In: $f \in R$, $B \subset R$
Out: a reduction of f w.r.t B

$f' := f$
WHILE $\exists g \in B$ such that $\mathrm{lm}(f')$ is divisible by $\mathrm{lm}(g)$; DO

$$f' := f' - \frac{\mathrm{lt}(f')}{\mathrm{lt}(g)} \cdot g$$

RETURN f'

Let R be a Gröbner-friendly algebra (think: $R = K[x_1, \ldots, x_n]$).

Definition

Given a fixed *admissible* monomial ordering, a polynomial $f \in R$ has

- initial monomial $\mathrm{lm}(f)$;
- initial coefficient $\mathrm{lc}(f)$;
- initial term $\mathrm{lt}(f) = \mathrm{lc}(f)\,\mathrm{lm}(f)$.

Algorithm $REDUCE(f, B)$

In: $f \in R$, $B \subset R$
Out: a reduction of f w.r.t B

$f' := f$
WHILE $\exists g \in B$ such that $\mathrm{lm}(f')$ is divisible by $\mathrm{lm}(g)$; DO

$$f' := f' - \frac{\mathrm{lt}(f')}{\mathrm{lt}(g)} \cdot g$$

RETURN f'

Let R be a Gröbner-friendly algebra (think: $R = K[x_1, \ldots, x_n]$).

Definition

Given a fixed *admissible* monomial ordering, a polynomial $f \in R$ has

- initial monomial $\mathrm{lm}(f)$;
- initial coefficient $\mathrm{lc}(f)$;
- initial term $\mathrm{lt}(f) = \mathrm{lc}(f)\,\mathrm{lm}(f)$.

Algorithm $REDUCE(f, B)$

In: $f \in R$, $B \subset R$
Out: a reduction of f w.r.t B

$f' := f$
WHILE $\exists g \in B$ such that $\mathrm{lm}(f')$ is divisible by $\mathrm{lm}(g)$; DO

$$f' := f' - \frac{\mathrm{lt}(f')}{\mathrm{lt}(g)} \cdot g$$

RETURN f'

Let R be a Gröbner-friendly algebra (think: $R = K[x_1, \ldots, x_n]$).

Definition

Given a fixed *admissible* monomial ordering, a polynomial $f \in R$ has

- initial monomial $\mathrm{lm}(f)$;
- initial coefficient $\mathrm{lc}(f)$;
- initial term $\mathrm{lt}(f) = \mathrm{lc}(f)\,\mathrm{lm}(f)$.

Algorithm $REDUCE(f, B)$

In: $f \in R$, $B \subset R$
Out: a reduction of f w.r.t B

$f' := f$
WHILE $\exists g \in B$ such that $\mathrm{lm}(f')$ is divisible by $\mathrm{lm}(g)$; DO

$$f' := f' - \frac{\mathrm{lt}(f')}{\mathrm{lt}(g)} \cdot g$$

RETURN f'

Let R be a Gröbner-friendly algebra (think: $R = K[x_1, \dots, x_n]$).

Definition

Given a fixed *admissible* monomial ordering, a polynomial $f \in R$ has

- initial monomial $\mathrm{lm}(f)$;
- initial coefficient $\mathrm{lc}(f)$;
- initial term $\mathrm{lt}(f) = \mathrm{lc}(f)\,\mathrm{lm}(f)$.

Algorithm $REDUCE(f, B)$

In: $f \in R$, $B \subset R$
Out: a reduction of f w.r.t B

$f' := f$
WHILE $\exists g \in B$ such that $\mathrm{lm}(f')$ is divisible by $\mathrm{lm}(g)$; DO

$$f' := f' - \frac{\mathrm{lt}(f')}{\mathrm{lt}(g)} \cdot g$$

RETURN f'

Let $L(f,g) = \mathrm{lcm}(\mathrm{lm}(f), \mathrm{lm}(g))$.

Definition (s-polynomial of f and g)

$$sPoly(f,g) = \mathrm{lc}(g)\frac{L(f,g)}{\mathrm{lm}(f)}f - \mathrm{lc}(f)\frac{L(f,g)}{\mathrm{lm}(g)}g.$$

Definition

A set $G \subset R$ is a Gröbner basis of a left ideal $I \subset R$ if $I = R \cdot G$ and

$$\mathrm{gr}(R) \cdot \{LM(f) | f \in I\} = \mathrm{gr}(R) \cdot \{LM(g) | g \in G\},$$

where $\mathrm{gr}(R)$ is the graded ring associated to R.

Buchberger criterion

A set $G \subset R$ is a Gröbner basis if $REDUCE(sPoly(f,g), G) = 0$ for all $f, g \in G$.

Let $L(f,g) = \mathrm{lcm}(\mathrm{lm}(f), \mathrm{lm}(g))$.

Definition (s-polynomial of f and g)

$$sPoly(f,g) = \mathrm{lc}(g)\frac{L(f,g)}{\mathrm{lm}(f)}f - \mathrm{lc}(f)\frac{L(f,g)}{\mathrm{lm}(g)}g.$$

Definition

A set $G \subset R$ is a Gröbner basis of a left ideal $I \subset R$ if $I = R \cdot G$ and

$$\mathrm{gr}(R) \cdot \{LM(f) | f \in I\} = \mathrm{gr}(R) \cdot \{LM(g) | g \in G\},$$

where $\mathrm{gr}(R)$ is the graded ring associated to R.

Buchberger criterion

A set $G \subset R$ is a Gröbner basis if $REDUCE(sPoly(f,g), G) = 0$ for all $f, g \in G$.

Let $L(f,g) = \mathrm{lcm}(\mathrm{lm}(f), \mathrm{lm}(g))$.

Definition (s-polynomial of f and g)

$$sPoly(f,g) = \mathrm{lc}(g)\frac{L(f,g)}{\mathrm{lm}(f)}f - \mathrm{lc}(f)\frac{L(f,g)}{\mathrm{lm}(g)}g.$$

Definition

A set $G \subset R$ is a Gröbner basis of a left ideal $I \subset R$ if $I = R \cdot G$ and

$$\mathrm{gr}(R) \cdot \{LM(f) | f \in I\} = \mathrm{gr}(R) \cdot \{LM(g) | g \in G\},$$

where $\mathrm{gr}(R)$ is the graded ring associated to R.

Buchberger criterion

A set $G \subset R$ is a Gröbner basis if $REDUCE(sPoly(f,g), G) = 0$ for all $f, g \in G$.

Buchberger algorithm

Given a generating set B of an ideal of R, algorithm $BUCHBERGER(B)$ computes a Gröbner basis G:

$G := B$
$S := \{(f_1, f_2) | f_1, f_2 \in B\}$ // queue of s-pairs
WHILE $S \neq \varnothing$; DO
  Pick $(f_1, f_2) \in S$, $S := S \setminus \{(f_1, f_2)\}$
  $g := REDUCE(sPoly(f_1, f_2), G)$
  IF $g \neq 0$
    THEN $S := S \cup \{(f, g) | f \in G\}$
      $G := G \cup \{g\}$
END WHILE
RETURN G

In the Weyl algebra...

- the basic version works;
- improved (Gebauer, Möller) version needs modifications.

Buchberger algorithm

Given a generating set B of an ideal of R, algorithm $BUCHBERGER(B)$ computes a Gröbner basis G:

$G := B$
$S := \{(f_1, f_2) | f_1, f_2 \in B\}$ // queue of s-pairs
WHILE $S \neq \varnothing$; DO
    Pick $(f_1, f_2) \in S$, $S := S \setminus \{(f_1, f_2)\}$
    $g := REDUCE(sPoly(f_1, f_2), G)$
    IF $g \neq 0$
        THEN $S := S \cup \{(f, g) | f \in G\}$
                $G := G \cup \{g\}$
END WHILE
RETURN G

In the Weyl algebra...

- the basic version works;
- improved (Gebauer, Möller) version needs modifications.

Buchberger algorithm

Given a generating set B of an ideal of R, algorithm $BUCHBERGER(B)$ computes a Gröbner basis G:

$G := B$
$S := \{(f_1, f_2) | f_1, f_2 \in B\}$ // queue of s-pairs
WHILE $S \neq \varnothing$; DO
  Pick $(f_1, f_2) \in S$, $S := S \setminus \{(f_1, f_2)\}$
  $g := REDUCE(sPoly(f_1, f_2), G)$
  IF $g \neq 0$
    THEN $S := S \cup \{(f, g) | f \in G\}$
      $G := G \cup \{g\}$
END WHILE
RETURN G

In the Weyl algebra...

- the basic version works;
- improved (Gebauer, Möller) version needs modifications.

Buchberger algorithm

Given a generating set B of an ideal of R, algorithm $BUCHBERGER(B)$ computes a Gröbner basis G:

$G := B$
$S := \{(f_1, f_2) | f_1, f_2 \in B\}$ // queue of s-pairs
WHILE $S \neq \varnothing$; DO
    Pick $(f_1, f_2) \in S$, $S := S \setminus \{(f_1, f_2)\}$
    $g := REDUCE(sPoly(f_1, f_2), G)$
    IF $g \neq 0$
        THEN $S := S \cup \{(f, g) | f \in G\}$
            $G := G \cup \{g\}$
END WHILE
RETURN G

In the Weyl algebra...

- the basic version works;
- improved (Gebauer, Möller) version needs modifications.

Prior work on parallel computation of Gröbner bases

- Buchberger (1987)
- Bündgen - Göbel - Küchlin (1994)
- Chakrabarti - Yelick (1994)
- Faugére (1994)
- Siegl (1994)
- Sawada - Terasaki - Aiba (1994)
- Attardi - Traverso (1996)
- Amrhein - Gloor - Kuchlin (1996)
- Sato - Suzuki (1999,2000)
- Gerdt - Yanovich (2005)

Prior work on parallel computation of Gröbner bases

- Buchberger (1987)
- Bündgen - Göbel - Küchlin (1994)
- Chakrabarti - Yelick (1994)
- Faugére (1994)
- Siegl (1994)
- Sawada - Terasaki - Aiba (1994)
- Attardi - Traverso (1996)
- Amrhein - Gloor - Kuchlin (1996)
- Sato - Suzuki (1999,2000)
- Gerdt - Yanovich (2005)

Prior work on parallel computation of Gröbner bases

- Buchberger (1987)
- Bündgen - Göbel - Küchlin (1994)
- Chakrabarti - Yelick (1994)
- Faugére (1994)
- Siegl (1994)
- Sawada - Terasaki - Aiba (1994)
- Attardi - Traverso (1996)
- Amrhein - Gloor - Kuchlin (1996)
- Sato - Suzuki (1999,2000)
- Gerdt - Yanovich (2005)

Prior work on parallel computation of Gröbner bases

- Buchberger (1987)
- Bündgen - Göbel - Küchlin (1994)
- Chakrabarti - Yelick (1994)
- Faugére (1994)
- Siegl (1994)
- Sawada - Terasaki - Aiba (1994)
- Attardi - Traverso (1996)
- Amrhein - Gloor - Kuchlin (1996)
- Sato - Suzuki (1999,2000)
- Gerdt - Yanovich (2005)

Prior work on parallel computation of Gröbner bases

- Buchberger (1987)
- Bündgen - Göbel - Küchlin (1994)
- Chakrabarti - Yelick (1994)
- Faugére (1994)
- Siegl (1994)
- Sawada - Terasaki - Aiba (1994)
- Attardi - Traverso (1996)
- Amrhein - Gloor - Kuchlin (1996)
- Sato - Suzuki (1999,2000)
- Gerdt - Yanovich (2005)

Prior work on parallel computation of Gröbner bases

- Buchberger (1987)
- Bündgen - Göbel - Küchlin (1994)
- Chakrabarti - Yelick (1994)
- Faugére (1994)
- Siegl (1994)
- Sawada - Terasaki - Aiba (1994)
- Attardi - Traverso (1996)
- Amrhein - Gloor - Kuchlin (1996)
- Sato - Suzuki (1999,2000)
- Gerdt - Yanovich (2005)

Prior work on parallel computation of Gröbner bases

- Buchberger (1987)
- Bündgen - Göbel - Küchlin (1994)
- Chakrabarti - Yelick (1994)
- Faugére (1994)
- Siegl (1994)
- Sawada - Terasaki - Aiba (1994)
- Attardi - Traverso (1996)
- Amrhein - Gloor - Kuchlin (1996)
- Sato - Suzuki (1999,2000)
- Gerdt - Yanovich (2005)

Prior work on parallel computation of Gröbner bases

- Buchberger (1987)
- Bündgen - Göbel - Küchlin (1994)
- Chakrabarti - Yelick (1994)
- Faugére (1994)
- Siegl (1994)
- Sawada - Terasaki - Aiba (1994)
- Attardi - Traverso (1996)
- Amrhein - Gloor - Kuchlin (1996)
- Sato - Suzuki (1999,2000)
- Gerdt - Yanovich (2005)

Prior work on parallel computation of Gröbner bases

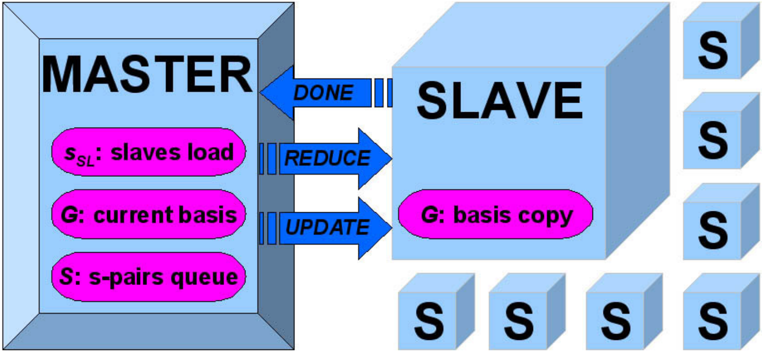
- Buchberger (1987)
- Bündgen - Göbel - Küchlin (1994)
- Chakrabarti - Yelick (1994)
- Faugére (1994)
- Siegl (1994)
- Sawada - Terasaki - Aiba (1994)
- Attardi - Traverso (1996)
- Amrhein - Gloor - Kuchlin (1996)
- Sato - Suzuki (1999,2000)
- Gerdt - Yanovich (2005)

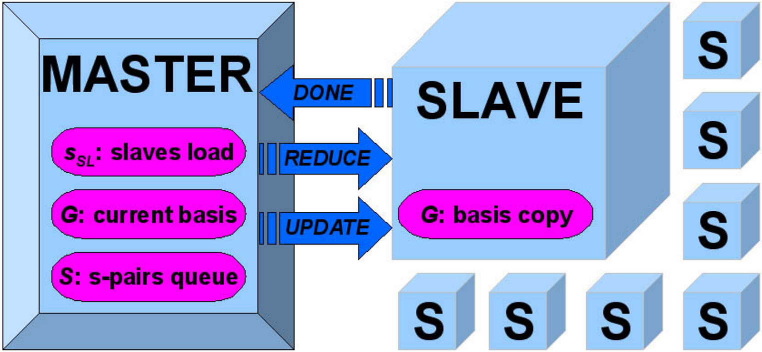
Prior work on parallel computation of Gröbner bases

- Buchberger (1987)
- Bündgen - Göbel - Küchlin (1994)
- Chakrabarti - Yelick (1994)
- Faugére (1994)
- Siegl (1994)
- Sawada - Terasaki - Aiba (1994)
- Attardi - Traverso (1996)
- Amrhein - Gloor - Kuchlin (1996)
- Sato - Suzuki (1999,2000)
- Gerdt - Yanovich (2005)

Prior work on parallel computation of Gröbner bases

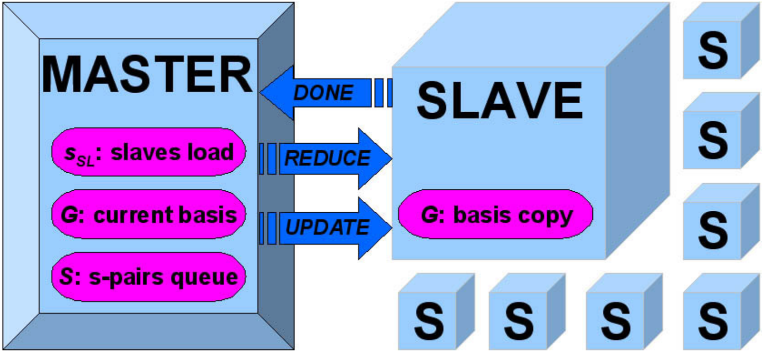
- Buchberger (1987)
- Bündgen - Göbel - Küchlin (1994)
- Chakrabarti - Yelick (1994)
- Faugére (1994)
- Siegl (1994)
- Sawada - Terasaki - Aiba (1994)
- Attardi - Traverso (1996)
- Amrhein - Gloor - Kuchlin (1996)
- Sato - Suzuki (1999,2000)
- Gerdt - Yanovich (2005)


MASTER
s_SL: slaves load
G: current basis
S: s-pairs queue
DONE
REDUCE
UPDATE
SLAVE
G: basis copy
S
S
S
S
S
S
S



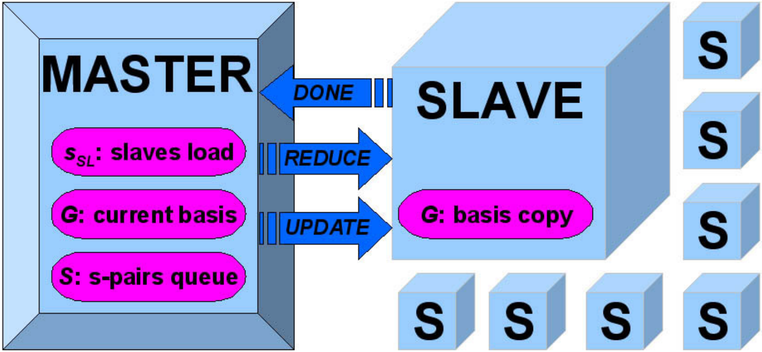
- maintains an intermediate basis G and the queue of s-pairs S;
- distributes orders to Slaves;
- collects results and updates G and S.

- stores a local basis G;
- receives orders from Master and send back the results;
- receives updates for G.



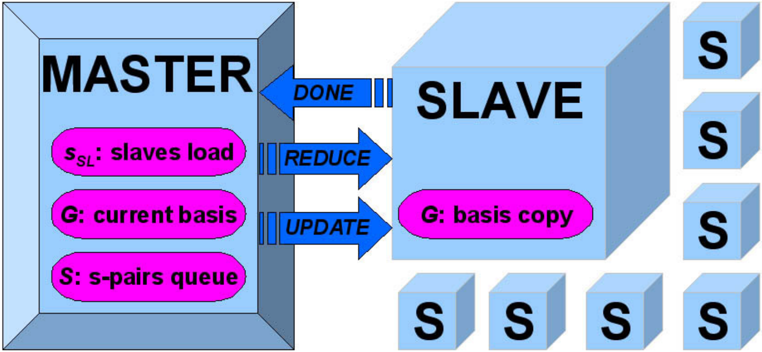
- maintains an intermediate basis G and the queue of s-pairs S;
- distributes orders to Slaves;
- collects results and updates G and S.

- stores a local basis G;
- receives orders from Master and send back the results;
- receives updates for G.



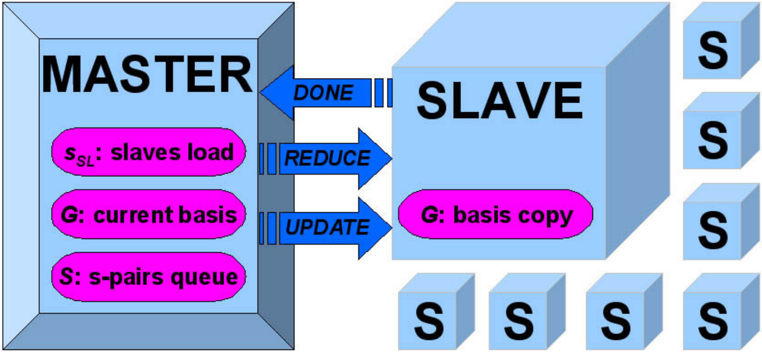
- maintains an intermediate basis G and the queue of s-pairs S;
- distributes orders to Slaves;
- collects results and updates G and S.

- stores a local basis G;
- receives orders from Master and send back the results;
- receives updates for G.



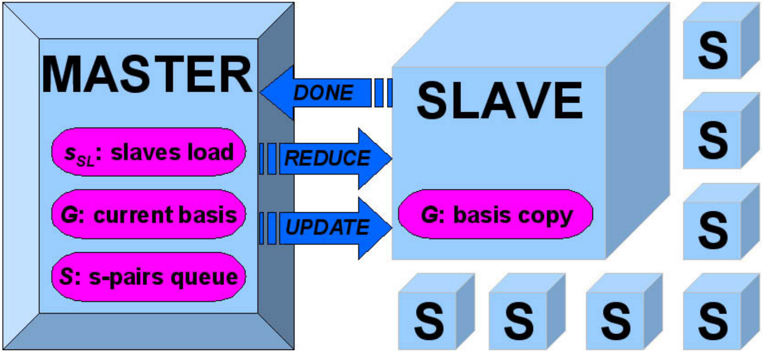
- maintains an intermediate basis G and the queue of s-pairs S;
- distributes orders to Slaves;
- collects results and updates G and S.

- stores a local basis G;
- receives orders from Master and send back the results;
- receives updates for G.



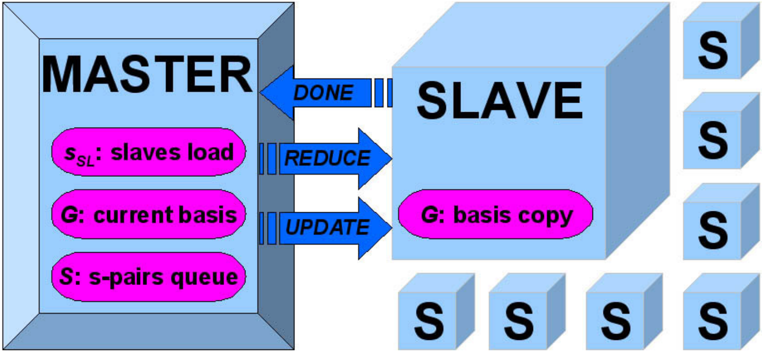
- maintains an intermediate basis G and the queue of s-pairs S;
- distributes orders to Slaves;
- collects results and updates G and S.

- stores a local basis G;
- receives orders from Master and send back the results;
- receives updates for G.



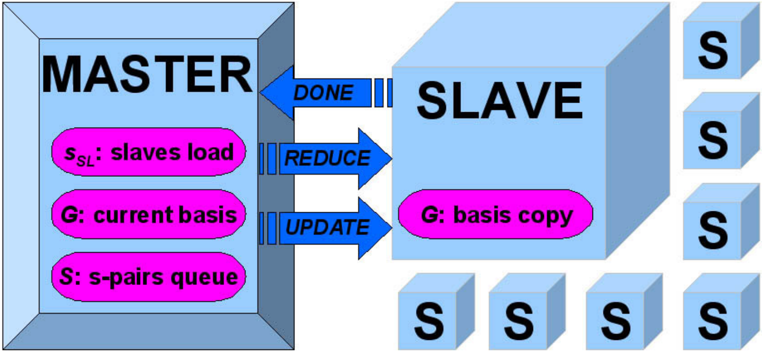
- maintains an intermediate basis G and the queue of s-pairs S;
- distributes orders to Slaves;
- collects results and updates G and S.

- stores a local basis G;
- receives orders from Master and send back the results;
- receives updates for G.



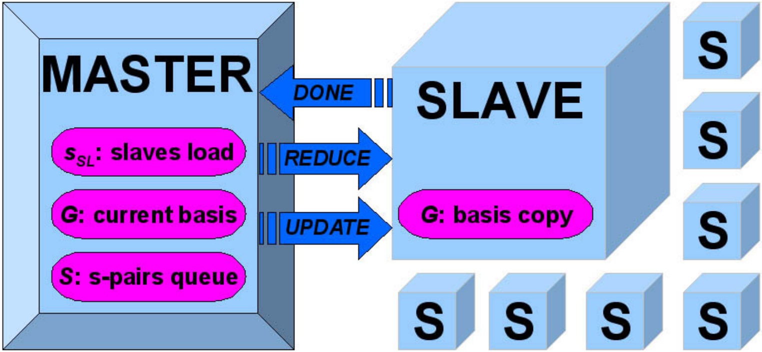
- maintains an intermediate basis G and the queue of s-pairs S;
- distributes orders to Slaves;
- collects results and updates G and S.

- stores a local basis G;
- receives orders from Master and send back the results;
- receives updates for G.



- maintains an intermediate basis G and the queue of s-pairs S;
- distributes orders to Slaves;
- collects results and updates G and S.

- stores a local basis G;
- receives orders from Master and send back the results;
- receives updates for G.



- maintains an intermediate basis G and the queue of s-pairs S;
- distributes orders to Slaves;
- collects results and updates G and S.

- stores a local basis G;
- receives orders from Master and send back the results;
- receives updates for G.

Key point: The order of s-pairs the same as in the serial algorithm. The strategies used for s-pair selection are preserved.

Implementation: C++ with MPI

- implemented from scratch in C++;
- uses MPI for communications;
- tested on clusters in the Minnesota Supercomputing Institute and NCSA.

Simulation of parallel computation

Assumptions:

- operations performed by Master are instantaneous;
- time for sending a package from one node to another depends linearly on its size.

Key point: The order of s-pairs the same as in the serial algorithm. The strategies used for s-pair selection are preserved.

Implementation: C++ with MPI

- implemented from scratch in C++;
- uses MPI for communications;
- tested on clusters in the Minnesota Supercomputing Institute and NCSA.

Simulation of parallel computation

Assumptions:

- operations performed by Master are instantaneous;
- time for sending a package from one node to another depends linearly on its size.

Key point: The order of s-pairs the same as in the serial algorithm. The strategies used for s-pair selection are preserved.

Implementation: C++ with MPI

- implemented from scratch in C++;
- uses MPI for communications;
- tested on clusters in the Minnesota Supercomputing Institute and NCSA.

Simulation of parallel computation

Assumptions:

- operations performed by Master are instantaneous;
- time for sending a package from one node to another depends linearly on its size.

Key point: The order of s-pairs the same as in the serial algorithm. The strategies used for s-pair selection are preserved.

Implementation: C++ with MPI

- implemented from scratch in C++;
- uses MPI for communications;
- tested on clusters in the Minnesota Supercomputing Institute and NCSA.

Simulation of parallel computation

Assumptions:

- operations performed by Master are instantaneous;
- time for sending a package from one node to another depends linearly on its size.

Key point: The order of s-pairs the same as in the serial algorithm. The strategies used for s-pair selection are preserved.

Implementation: C++ with MPI

- implemented from scratch in C++;
- uses MPI for communications;
- tested on clusters in the Minnesota Supercomputing Institute and NCSA.

Simulation of parallel computation

Assumptions:

- operations performed by Master are instantaneous;
- time for sending a package from one node to another depends linearly on its size.

Key point: The order of s-pairs the same as in the serial algorithm. The strategies used for s-pair selection are preserved.

Implementation: C++ with MPI

- implemented from scratch in C++;
- uses MPI for communications;
- tested on clusters in the Minnesota Supercomputing Institute and NCSA.

Simulation of parallel computation

Assumptions:

- operations performed by Master are instantaneous;
- time for sending a package from one node to another depends linearly on its size.

Key point: The order of s-pairs the same as in the serial algorithm. The strategies used for s-pair selection are preserved.

Implementation: C++ with MPI

- implemented from scratch in C++;
- uses MPI for communications;
- tested on clusters in the Minnesota Supercomputing Institute and NCSA.

Simulation of parallel computation

Assumptions:

- operations performed by Master are instantaneous;
- time for sending a package from one node to another depends linearly on its size.

Key point: The order of s-pairs the same as in the serial algorithm. The strategies used for s-pair selection are preserved.

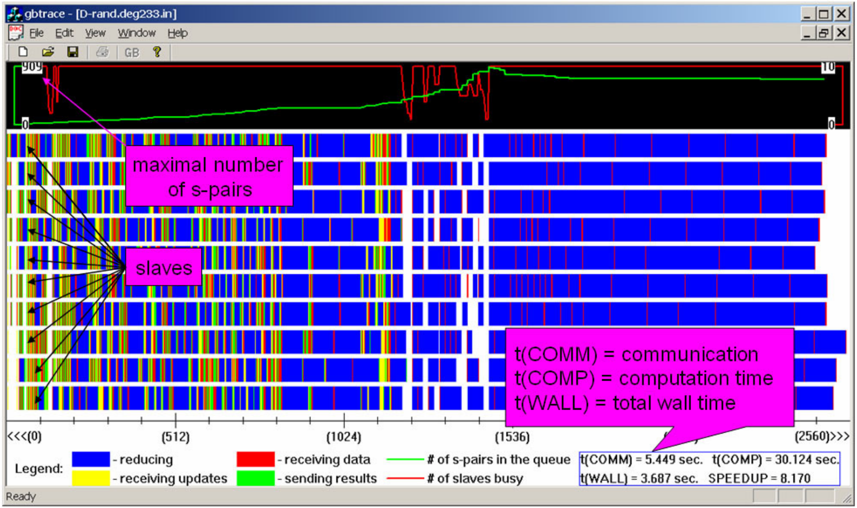
Implementation: C++ with MPI

- implemented from scratch in C++;
- uses MPI for communications;
- tested on clusters in the Minnesota Supercomputing Institute and NCSA.

Simulation of parallel computation

Assumptions:

- operations performed by Master are instantaneous;
- time for sending a package from one node to another depends linearly on its size.


gbtrace - [D-rand.deg233.in]
File Edit View Window Help
909
10
0
0
maximal number of s-pairs
slaves
t(COMM) = communication
t(COMP) = computation time
t(WALL) = total wall time
<<<(0)
(512)
(1024)
(1536)
(2560)>>>
Legend:
- reducing
- receiving updates
- receiving data
- sending results
# of s-pairs in the queue
# of slaves busy
t(COMM) = 5.449 sec. t(COMP) = 30.124 sec.
t(WALL) = 3.687 sec. SPEEDUP = 8.170
Ready

Faugére's F_4:

- can be adapted for Gröbner-friendly algebras;
- implementations: Noro (risa/asir) for Weyl algebra, Pearce - Monagan (Maple) Ore algebras;
- loss of sparsity: multiplication of an operator by monomial increases the number of terms.

Example

Let $D = A_2 = K\langle x_1, x_2, \partial_1, \partial_2\rangle$, $f_1 = x_1^2\partial_2^3$, $f_2 = x_2^3\partial_1^2$. F_4 starts computing a Gröbner basis of ideal $D \cdot \{f_1, f_2\}$ with the matrix

$$\begin{array}{cc} & \begin{array}{cc} x_1^2\partial_2^3 & x_2^3\partial_1^2 \end{array} \\ \begin{array}{c} f_1 \\ f_2 \end{array} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array}$$

Faugére's F_4:

- can be adapted for Gröbner-friendly algebras;
- implementations: Noro (risa/asir) for Weyl algebra, Pearce - Monagan (Maple) Ore algebras;
- loss of sparsity: multiplication of an operator by monomial increases the number of terms.

Example

Let $D = A_2 = K\langle x_1, x_2, \partial_1, \partial_2\rangle$, $f_1 = x_1^2\partial_2^3$, $f_2 = x_2^3\partial_1^2$. F_4 starts computing a Gröbner basis of ideal $D \cdot \{f_1, f_2\}$ with the matrix

$$\begin{array}{cc} & \begin{array}{cc} x_1^2\partial_2^3 & x_2^3\partial_1^2 \end{array} \\ \begin{array}{c} f_1 \\ f_2 \end{array} & \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right] \end{array}$$

Faugére's F_4:

- can be adapted for Gröbner-friendly algebras;
- implementations: Noro (risa/asir) for Weyl algebra, Pearce - Monagan (Maple) Ore algebras;
- loss of sparsity: multiplication of an operator by monomial increases the number of terms.

Example

Let $D = A_2 = K\langle x_1, x_2, \partial_1, \partial_2\rangle$, $f_1 = x_1^2\partial_2^3$, $f_2 = x_2^3\partial_1^2$. F_4 starts computing a Gröbner basis of ideal $D \cdot \{f_1, f_2\}$ with the matrix

$$\begin{array}{cc} & \begin{array}{cc} x_1^2\partial_2^3 & x_2^3\partial_1^2 \end{array} \\ \begin{array}{c} f_1 \\ f_2 \end{array} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array}$$

Faugére's F_4:

- can be adapted for Gröbner-friendly algebras;
- implementations: Noro (risa/asir) for Weyl algebra, Pearce - Monagan (Maple) Ore algebras;
- loss of sparsity: multiplication of an operator by monomial increases the number of terms.

Example

Let $D = A_2 = K\langle x_1, x_2, \partial_1, \partial_2\rangle$, $f_1 = x_1^2\partial_2^3$, $f_2 = x_2^3\partial_1^2$. F_4 starts computing a Gröbner basis of ideal $D \cdot \{f_1, f_2\}$ with the matrix

$$\begin{array}{cc} & \begin{array}{cc} x_1^2\partial_2^3 & x_2^3\partial_1^2 \end{array} \\ \begin{array}{c} f_1 \\ f_2 \end{array} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array}$$

The second step builds the following matrix:

$$
\begin{array}{c|cccccccc|}
 & x_1^2x_2^3\partial_1^2\partial_2^3 & x_1^2x_2^2\partial_1^2\partial_2^2 & x_1x_2^3\partial_1\partial_2^3 & x_1^2x_2\partial_1^2\partial_2 & x_2^3\partial_2^3 & x_2^3\partial_1^2 & x_1^2\partial_2^3 & x_1^2\partial_1^2 \\
\hline
f_1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
f_2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
f_1f_2 & 1 & 9 & 0 & 18 & 0 & 0 & 0 & 1 \\
f_2f_1 & 1 & 0 & 4 & 0 & 1 & 0 & 0 & 0
\end{array}
$$

Help!

- Structured Gaussian elimination? ... (over a finite field) ...
- ... in parallel?

The second step builds the following matrix:

$$
\begin{array}{c}
 \\ f_1 \\ f_2 \\ f_1 f_2 \\ f_2 f_1
\end{array}
\begin{array}{c}
\begin{array}{cccccccc}
x_1^2x_2^3\partial_1^2\partial_2^3 & x_1^2x_2^2\partial_1^2\partial_2^2 & x_1x_2^3\partial_1\partial_2^3 & x_1^2x_2\partial_1^2\partial_2 & x_2^3\partial_2^3 & x_2^3\partial_1^2 & x_1^2\partial_2^3 & x_1^2\partial_1^2
\end{array} \\
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 9 & 0 & 18 & 0 & 0 & 0 & 1 \\
1 & 0 & 4 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$

Help!

- Structured Gaussian elimination? ... (over a finite field) ...
- ... in parallel?

The second step builds the following matrix:

$$
\begin{array}{c}
 \\
f_1 \\
f_2 \\
f_1 f_2 \\
f_2 f_1
\end{array}
\begin{array}{c}
\begin{array}{cccccccc}
x_1^2x_2^3\partial_1^2\partial_2^3 & x_1^2x_2^2\partial_1^2\partial_2^2 & x_1x_2^3\partial_1\partial_2^3 & x_1^2x_2\partial_1^2\partial_2 & x_2^3\partial_2^3 & x_2^3\partial_1^2 & x_1^2\partial_2^3 & x_1^2\partial_1^2
\end{array} \\
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 9 & 0 & 18 & 0 & 0 & 0 & 1 \\
1 & 0 & 4 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$

Help!

- Structured Gaussian elimination? ... (over a finite field) ...
- ... in parallel?

Conclusion

- Parallel versions of Buchberger's algorithm can produce limited speedups;
- Our coarse-grain approach exhibits better speedups in the noncommutative algebra than in the (commutative) polynomial rings on "interesting" problems of similar size.
- It does make sense to use 128 nodes on this problem!

Future

- From theory to practice: a practically efficient parallel implementation is needed;
- Faugére's F_4 algorithm results in the loss of sparsity in the intermediate computation...
- ... however, it still feasible and its parallel version could be constructed;
- Agreeing on the test/benchmark set.

Conclusion

- Parallel versions of Buchberger's algorithm can produce limited speedups;
- Our coarse-grain approach exhibits better speedups in the noncommutative algebra than in the (commutative) polynomial rings on "interesting" problems of similar size.
- It does make sense to use 128 nodes on this problem!

Future

- From theory to practice: a practically efficient parallel implementation is needed;
- Faugére's F_4 algorithm results in the loss of sparsity in the intermediate computation...
- ... however, it still feasible and its parallel version could be constructed;
- Agreeing on the test/benchmark set.

Conclusion

- Parallel versions of Buchberger's algorithm can produce limited speedups;
- Our coarse-grain approach exhibits better speedups in the noncommutative algebra than in the (commutative) polynomial rings on "interesting" problems of similar size.
- It does make sense to use 128 nodes on this problem!

Future

- From theory to practice: a practically efficient parallel implementation is needed;
- Faugére's F_4 algorithm results in the loss of sparsity in the intermediate computation...
- ... however, it still feasible and its parallel version could be constructed;
- Agreeing on the test/benchmark set.

Conclusion

- Parallel versions of Buchberger's algorithm can produce limited speedups;
- Our coarse-grain approach exhibits better speedups in the noncommutative algebra than in the (commutative) polynomial rings on "interesting" problems of similar size.
- It does make sense to use 128 nodes on this problem!

Future

- From theory to practice: a practically efficient parallel implementation is needed;
- Faugére's F_4 algorithm results in the loss of sparsity in the intermediate computation...
- ... however, it still feasible and its parallel version could be constructed;
- Agreeing on the test/benchmark set.

Conclusion

- Parallel versions of Buchberger's algorithm can produce limited speedups;
- Our coarse-grain approach exhibits better speedups in the noncommutative algebra than in the (commutative) polynomial rings on "interesting" problems of similar size.
- It does make sense to use 128 nodes on this problem!

Future

- From theory to practice: a practically efficient parallel implementation is needed;
- Faugére's F_4 algorithm results in the loss of sparsity in the intermediate computation...
- ... however, it still feasible and its parallel version could be constructed;
- Agreeing on the test/benchmark set.

Conclusion

- Parallel versions of Buchberger's algorithm can produce limited speedups;
- Our coarse-grain approach exhibits better speedups in the noncommutative algebra than in the (commutative) polynomial rings on "interesting" problems of similar size.
- It does make sense to use 128 nodes on this problem!

Future

- From theory to practice: a practically efficient parallel implementation is needed;
- Faugére's F_4 algorithm results in the loss of sparsity in the intermediate computation...
- ... however, it still feasible and its parallel version could be constructed;
- Agreeing on the test/benchmark set.

Conclusion

- Parallel versions of Buchberger's algorithm can produce limited speedups;
- Our coarse-grain approach exhibits better speedups in the noncommutative algebra than in the (commutative) polynomial rings on "interesting" problems of similar size.
- It does make sense to use 128 nodes on this problem!

Future

- From theory to practice: a practically efficient parallel implementation is needed;
- Faugére's F_4 algorithm results in the loss of sparsity in the intermediate computation...
- ... however, it still feasible and its parallel version could be constructed;
- Agreeing on the test/benchmark set.

Conclusion

- Parallel versions of Buchberger's algorithm can produce limited speedups;
- Our coarse-grain approach exhibits better speedups in the noncommutative algebra than in the (commutative) polynomial rings on "interesting" problems of similar size.
- It does make sense to use 128 nodes on this problem!

Future

- From theory to practice: a practically efficient parallel implementation is needed;
- Faugére's F_4 algorithm results in the loss of sparsity in the intermediate computation...
- ... however, it still feasible and its parallel version could be constructed;
- Agreeing on the test/benchmark set.

Conclusion

- Parallel versions of Buchberger's algorithm can produce limited speedups;
- Our coarse-grain approach exhibits better speedups in the noncommutative algebra than in the (commutative) polynomial rings on "interesting" problems of similar size.
- It does make sense to use 128 nodes on this problem!

Future

- From theory to practice: a practically efficient parallel implementation is needed;
- Faugére's F_4 algorithm results in the loss of sparsity in the intermediate computation...
- ... however, it still feasible and its parallel version could be constructed;
- Agreeing on the test/benchmark set.