Multiplication of polynomials

M. Gastineau

IMCCE - Observatoire de Paris - CNRS UMR8028
77, avenue Denfert Rochereau
75014 PARIS
FRANCE

gastineau@imcce.fr
Different products

- Full
  - All terms are computed

- Truncated in the partial or total degree of the variables

- Special truncation to select terms satisfying a rule
Available methods

- **Naive method**
  - efficient for low degree or for sparse polynomials

- **Karatsuba’s algorithm**
  - efficient for intermediate degree and dense polynomials
  - reduce the number of multiplications

- **FFT method**
  - efficient only for large degree
Naive multiplication

$$A(x) = \sum_{i=d_{a\, min}}^{d_{a\, max}} a_i x^i \text{ and } B(x) = \sum_{i=d_{b\, min}}^{d_{b\, max}} b_i x^i$$

- Perform the multiplication of all terms

$$C(x) = \sum_{k=d_{a\, min} + d_{b\, min}}^{d_{a\, max} + d_{b\, max}} c_k x^k \text{ with } c_k = \sum_{i+j=k} a_i b_j$$

- if $A$, $B$ and $C$, have $r$, $s$ and $t$ terms
  - $rs$ multiplications and $rs-t$ additions
  - complexity : $O(rs)$
Algorithm 1: Compute the full product of univariate polynomials $A$ and $B$ represented with a dense vector

**Input:** $A$: polynomial $\{da_{min}, da_{max}, \text{array of coefficients } a\}$

**Input:** $B$: polynomial $\{db_{min}, db_{max}, \text{array of coefficients } b\}$

**Output:** $C$: polynomial $\{dc_{min}, dc_{max}, \text{array of coefficients } c\}$

\[
dc_{min} \leftarrow da_{min} + db_{min}
\]
\[
dc_{max} \leftarrow da_{max} + db_{max}
\]

$c$ ← create a polynomial with minimal degree $dc_{min}$ and maximal degree $dc_{max}$

for $k \leftarrow dc_{min}$ to $dc_{max}$ do
    $c[k] \leftarrow a[da_{min}] \times b[k - da_{min}]$
    for $j \leftarrow da_{min} + 1$ to $da_{max}$ do
        $c[k] \leftarrow c[k] + a[j] \times b[k - j]$
    end
end

return $C$
Function `mulfull(A,B)` Compute the full product of multivariate polynomials $A$ and $B$ represented with a recursive dense vector

**Input:** $A$: multivariate polynomial $\{da_{min}, da_{max}, \text{array of coefficients } a\}$

**Input:** $B$: multivariate polynomial $\{db_{min}, db_{max}, \text{array of coefficients } b\}$

**Output:** $C$: multivariate polynomial $\{dc_{min}, dc_{max}, \text{array of coefficients } c\}$

$dc_{min} \leftarrow da_{min} + db_{min}$

$dc_{max} \leftarrow da_{max} + db_{max}$

$C \leftarrow$ create a polynomial with minimal $dc_{min}$ and maximal $dc_{max}$ degree

for $k \leftarrow dc_{min}$ to $dc_{max}$ do

\[ c[k] \leftarrow \text{mulfull}(a[da_{min}], b[k - da_{min}]) \]

for $j \leftarrow da_{min} + 1$ to $da_{max}$ do

\[ \text{fmafull}(a[j], b[k - j], c[k]) \]

end

end

return $C$
Procedure \texttt{fmafull}(A,B,C) Compute the full fused multiplication-addition $C = C + A \times B$ with $A$, $B$ and $C$ multivariate polynomials represented as recursive dense vector

| Input: $A$: multivariate polynomial $\{da_{\text{min}}, da_{\text{max}}, \text{array of coefficients } a\}$ |
| Input: $B$: multivariate polynomial $\{db_{\text{min}}, db_{\text{max}}, \text{array of coefficients } b\}$ |
| Input: $C$: multivariate polynomial $\{dc_{\text{min}}, dc_{\text{max}}, \text{array of coefficients } c\}$ |
| Output: $C$: multivariate polynomial |

$newdc_{\text{min}} \leftarrow da_{\text{min}} + db_{\text{min}}$

$newdc_{\text{max}} \leftarrow da_{\text{max}} + db_{\text{max}}$

\textbf{if} $newdc_{\text{min}} < dc_{\text{min}}$ or $dc_{\text{max}} < newdc_{\text{max}}$ \textbf{then}

\hspace{1em} $dc_{\text{min}} \leftarrow \min(newdc_{\text{min}}, dc_{\text{min}})$

\hspace{1em} $dc_{\text{max}} \leftarrow \max(newdc_{\text{max}}, dc_{\text{max}})$

\hspace{1em} resize $C$

\textbf{end}

\textbf{for} $k \leftarrow dc_{\text{min}}$ \textbf{to} $dc_{\text{max}}$ \textbf{do}

\hspace{1em} $c[k] \leftarrow \text{mulfull}(a[da_{\text{min}}], b[k - da_{\text{min}}])$

\hspace{1em} \textbf{for} $j \leftarrow da_{\text{min}} + 1$ \textbf{to} $da_{\text{max}}$ \textbf{do}

\hspace{2em} \texttt{fmafull}(a[j], b[k - j], c[k])

\textbf{end}

\textbf{end}

\textbf{if} $c$ contains 0 at its beginning or at its end \textbf{then}

\hspace{1em} adjust $dc_{\text{min}}$

\hspace{1em} adjust $dc_{\text{max}}$

\hspace{1em} resize $C$

\textbf{end}
Function `mulfull(A,B)` Compute the full product of univariate polynomials $A$ and $B$ represented with a list

**Input:** $A$: polynomial \{ list of (coefficients $a$, degree $\delta_a$) \}

**Input:** $B$: polynomial \{ list of (coefficients $b$, degree $\delta_b$) \}

**Output:** $C$: polynomial \{ list of (coefficients $c$, degree $\delta_c$) \}

1. $C \leftarrow$ create a empty polynomial
2. **foreach** element in $A$ **do**
   1. $D \leftarrow$ create a empty polynomial
   2. **foreach** element in $B$ **do**
     1. add to the tail of $D$ an element $(a \times b, \delta_a + \delta_b)$
   3. $C \leftarrow C + D$
3. **end**
4. **return** $C$
Procedure fmafull(A,B,C) Compute the full fused multiplication-
addition $C = C + A \times B$ with $A, B$ and $C$ multivariate polynomials
represented as recursive list

**Input:** A: polynomial { list of (coefficients $a$, degree $\delta_a$) }
**Input:** B: polynomial { list of (coefficients $b$, degree $\delta_b$) }
**Input:** C: polynomial { list of (coefficients $c$, degree $\delta_c$) }

**Output:** C: polynomial { list of (coefficients $c$, degree $\delta_c$) }

iter $\leftarrow$ head of C

foreach element in A do

    // avoid to scan to C when the loop on B is finished
    iterb $\leftarrow$ iter

    foreach element in B do

        // find after iterb in C if the degree $\delta_a + \delta_b$ is present
        while current degree $\delta_c$ referenced by iterb $< \delta_a + \delta_b$ do
            iterb $\leftarrow$ next element after iterb
        end

        if $\delta_c = \delta_a + \delta_b$ then
            fmafull (a, b, c)
        end

        if $c = 0$ then remove the element referenced by iterb
        else
            insert an element ($\text{mulfull} (a, b), \delta_a + \delta_b$) just before iterb
        end

    if current element is the first element of B then
        iter $\leftarrow$ iterb
    end

end
Multiplication for flat vector 1/2

How to sort terms?

- search and shift operations too slow
- need an intermediate and adjustable storage: BURST TRIE
Multiplication for flat vector 2/2

- Burst tries
  - trie node = dense container
  - leaf node = sparse container

\[3 + 5z + 7z^3 + 11y + 9zy + 13zyx + 8z^2x^2 + 9x^4\]
Homogeneous block

\[ A = \sum_{\delta = d_{a_{\text{min}}}}^{d_{a_{\text{max}}}} B H_\delta(a) \quad , \quad B = \sum_{\delta = d_{b_{\text{min}}}}^{d_{b_{\text{max}}}} B H_\delta(b) \]

\[ a_i X_1^{d_1} X_2^{d_2} \cdots X_n^{d_n} \times b_j X_1^{d'_1} X_2^{d'_2} \cdots X_n^{d'_n} = a_i b_j X_1^{d_1+d'_1} X_2^{d_2+d'_2} \cdots X_n^{d_n+d'_n} \]

\[ C = A \times B = \sum_{\delta = d_{c_{\text{min}}}}^{d_{c_{\text{max}}}} B H_\delta(c) \]

with

\[ d_{c_{\text{min}}} = d_{a_{\text{min}}} + d_{b_{\text{min}}} \]
\[ d_{c_{\text{max}}} = d_{a_{\text{max}}} + d_{b_{\text{max}}} \]

\[ B H_\delta(c) = \sum_{i+j=\delta} B H_i(a) \times B H_j(b) \]
**Homogeneous block**

---

**Function** $fmafull(BH_\delta(a), BH_{\delta'}(b), BH_{\delta+\delta'}(c))$

Compute the full fused multiplication-addition

$$BH_{\delta+\delta'}(c) = BH_{\delta+\delta'}(c) + BH_\delta(a) \times BH_{\delta'}(b)$$

**Input:** $BH_\delta(a)$: homogeneous blocks \{ degree $\delta$, $a$ : array of $r$ coeff. \}

**Input:** $BH_{\delta'}(b)$: homogeneous blocks \{ degree $\delta'$, $b$ : array of $s$ coeff. \}

**Input:** $BH_{\delta+\delta'}(c)$: homog. blocks \{ degree $\delta + \delta'$, $c$ : array of $t$ coeff. \}

**Output:** $BH_{\delta+\delta'}(c)$: homog. blocks \{ degree $\delta + \delta'$, $c$ : array of $t$ coeff. \}

for $i \leftarrow 1$ to $r$ do
  for $j \leftarrow 1$ to $s$ do
    $l \leftarrow$ get location of the term in $BH_{\delta+\delta'}(c)$
    $BH_{\delta+\delta'}(c)[l] \leftarrow BH_{\delta+\delta'}(c)[l] + BH_\delta(a)[i] \times BH_{\delta'}(b)[j]$
  end
end
Homogeneous block using functions

**Function** \( f_{mafull}(BH_\delta(a), BH_\delta'(b), BH_\delta+\delta'(c)) \)

Compute the full fused multiplication-addition

\[
BH_\delta+\delta'(c) = BH_\delta+\delta'(c) + BH_\delta(a) \times BH_\delta'(b)
\]

using functions to compute location

**Input:** \( BH_\delta(a) \): homogeneous blocks \{ degree \( \delta \), \( a \) : array of \( r \) coeff. \}  
**Input:** \( BH_\delta'(b) \): homogeneous blocks \{ degree \( \delta' \), \( b \) : array of \( s \) coeff.s \}  
**Input:** \( BH_\delta+\delta'(c) \): homog. blocks \{ degree \( \delta+\delta' \), \( c \) : array of \( t \) coeff. \}  
**Output:** \( BH_\delta+\delta'(c) \): homog. blocks \{ degree \( \delta+\delta' \), \( c \) : array of \( t \) coeff. \}

for \( i \leftarrow 1 \) to \( r \) do

\( \text{expoa} \leftarrow \text{get array of exponents from the location } i \text{ in } BH_\delta \)

for \( j \leftarrow 1 \) to \( s \) do

\( \text{expob} \leftarrow \text{get array of exponents from the location } j \text{ in } BH_\delta' \)

\( \text{expoc} \leftarrow \text{expoa} + \text{expob} \)

\( l \leftarrow \text{get location of the term with exponents } \text{expoc} \text{ in } BH_\delta+\delta' \)

\( BH_\delta+\delta'(c)[l] \leftarrow BH_\delta+\delta'(c)[l] + BH_\delta(a)[i] \times BH_\delta'(b)[j] \)

end

end
Homogeneous block using addressing tables

Construction of the addressing table for the product of blocks in 3 variables with exponent tables of degree 1 and 2.

\[ T_{exp1} + T_{exp2} = T_{exp3} \]

\[
\begin{array}{c|c|c}
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
\hline
0 & 0 & 2 \\
0 & 1 & 1 \\
0 & 2 & 0 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
2 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c}
0 & 0 & 3 \\
0 & 1 & 2 \\
0 & 2 & 1 \\
0 & 3 & 0 \\
1 & 0 & 2 \\
1 & 1 & 1 \\
1 & 2 & 0 \\
2 & 0 & 1 \\
2 & 1 & 0 \\
3 & 0 & 0 \\
\end{array}
\]

\[ T_{addr1,2} \]

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 5 & 6 & 8 \\
2 & 3 & 4 & 6 & 7 & 9 \\
5 & 6 & 7 & 8 & 9 & 10 \\
\end{array}
\]

\[ T_{addr2,1} = t T_{addr1,2} \]
Execution time to build the tables of exponents

build the tables of exponents for homogeneous blocks in 10 variables up to the degree 20
Execution time to build the addressing tables

Product of two homogeneous blocks in 5 variables up to the total degree 40

Graph showing the execution time in seconds (y-axis) versus the number of multiplication operations (in Millions, x-axis) for initializing and after initialization.
Overhead to load the addressing tables from disk
Homogeneous blocks

\[ P(x, y, z) = 3 + 5z + 7z^3 + 11y + 9yz + 13xyz + 8x^2 z^2 + 9x^4 \]
Compacted homogeneous blocks

\[
\begin{array}{cccc}
\text{BHC}_0 & \text{BHC}_1 & \text{BHC}_2 & \text{BHC}_3 & \text{BHC}_4 \\
3 & 5 & 9 & 13 & 8 \\
11 & 1 & 2 & 6 & 10 \\
\end{array}
\]
Homogeneous block using addressing tables

**Function** \( fmafull( BH_\delta(a), BH_\delta'(b), Taddr_\delta,\delta', BH_\delta+\delta'(c)) \)
Compute the full fused multiplication-addition
\( BH_\delta+\delta'(c) = BH_\delta+\delta'(c) + BH_\delta(a) \times BH_\delta'(b) \) using the addressing table

**Input:**
- \( BH_\delta(a) \): homogeneous blocks \{ degree \( \delta \), \( a \) : array of \( r \) coeff. \}
- \( BH_\delta'(b) \): homogeneous blocks \{ degree \( \delta' \), \( b \) : array of \( s \) coeff. \}
- \( BH_\delta+\delta'(c) \): homog. blocks \{ degree \( \delta + \delta' \), \( c \) : array of \( t \) coeff. \}
- \( Taddr_\delta,\delta' \): addressing table of degree \( \delta, \delta' \)

**Output:** \( BH_\delta+\delta'(c) \): homog. blocks \{ degree \( \delta + \delta' \), \( c \) : array of \( t \) coefficients \}

\[
\text{for } i \leftarrow 1 \text{ to } r \text{ do}\n\quad \text{for } j \leftarrow 1 \text{ to } s \text{ do}\n\quad\quad l \leftarrow Taddr_\delta,\delta'[i,j]\n\quad\quad BH_\delta+\delta'(c)[l] \leftarrow BH_\delta+\delta'(c)[l] + BH_\delta(a)[i] \times BH_\delta'(b)[j]\n\quad\text{end}\n\quad\text{end}\n\]

Advanced School on Specific Algebraic Manipulators, 2007, © M. Gastineau, ASD/IMCCE/CNRS
Homogeneous block using addressing tables

**Function** fmafull($BH_\delta(a)$, $BH_{\delta'}(b)$, $Taddr_{\delta,\delta'}$, $BH_{\delta+\delta'}(c)$)

Compute the full fused multiplication-addition

$BH_{\delta+\delta'}(c) = BH_{\delta+\delta'}(c) + BH_\delta(a) \times BH_{\delta'}(b)$ using the addressing table

**Input:**
- $BH_\delta(a)$: homogeneous blocks \{ degree $\delta$, $a$: array of $r$ coeff. \}
- $BH_{\delta'}(b)$: homogeneous blocks \{ degree $\delta'$, $b$: array of $s$ coeff. \}
- $BH_{\delta+\delta'}(c)$: homog. blocks \{ degree $\delta + \delta'$, $c$: array of $t$ coeff. \}
- $Taddr_{\delta,\delta'}$: addressing table of degree $\delta, \delta'$

**Output:** $BH_{\delta+\delta'}(c)$: homog. blocks \{ degree $\delta + \delta'$, $c$: array of $t$ coefficients \}

for $i \leftarrow 1$ to $r$ do
  for $j \leftarrow 1$ to $s$ do
    $l \leftarrow Taddr_{\delta,\delta'}[BH_{C\delta}(a).index[i], BH_{C\delta'}(b).index[j]]$
    $BH_{\delta+\delta'}(c)[l] \leftarrow BH_{\delta+\delta'}(c)[l] + BH_\delta(a)[i] \times BH_{\delta'}(b)[j]$
  end
end
Homogeneous block using addressing tables

**Function** `fmafull(BH_δ(a), BH_{δ'}(b), Taddr_{δ,δ'}, BH_{δ+δ'}(c))`

Compute the full fused multiplication-addition

\[ BH_{δ+δ'}(c) = BH_{δ+δ'}(c) + BH_δ(a) \times BH_{δ'}(b) \] using the addressing table

**Input:**
- `BH_δ(a)`: homogeneous blocks \{ degree \( δ \), \( a \): array of \( r \) coeff. \}
- `BH_{δ'}(b)`: homogeneous blocks \{ degree \( δ' \), \( b \): array of \( s \) coeff. \}
- `BH_{δ+δ'}(c)`: homog. blocks \{ degree \( δ + δ' \), \( c \): array of \( t \) coeff. \}
- `Taddr_{δ,δ'}`: addressing table of degree \( δ, δ' \)

**Output:** `BH_{δ+δ'}(c)`: homog. blocks \{ degree \( δ + δ' \), \( c \): array of \( t \) coefficients \}

```plaintext
for i ← 1 to r do
    for j ← 1 to s do
        l ← Taddr_{δ,δ'}[i, j]
        BH_{δ+δ'}(c)[l] ← BH_{δ+δ'}(c)[l] + BH_δ(a)[i] \times BH_{δ'}(b)[j]
    end
end
```
Cache blocking technique

normal flow

BHδ(a)

BHδ(b)

flow with cache blocking

BHδ(a)

BHδ(b)
Cache blocking technique

Input: $BH_\delta(a)$: homogeneous blocks { degree $\delta$, $a$: array of $r$ coeff. }
Input: $BH_{\delta'}(b)$: homogeneous blocks { degree $\delta'$, $b$: array of $s$ coeff. }
Input: $Taddr_{\delta,\delta'}$: addressing table of degree $\delta, \delta'$
Input: $BH_{\delta+\delta'}(c)$: homog. blocks { degree $\delta + \delta'$, $c$: array of $t$ coeff. }
Input: $\text{chunksize}$: chunk size { arrays of two integers}
Output: $BH_{\delta+\delta'}(c)$: homog. blocks { degree $\delta + \delta'$, $c$: array of $t$ coeff. }

1. $\text{iteri} \leftarrow r/\text{chunksize}[1] \quad /\!*\text{ number of chunks for the loop } i \quad */$
2. $\text{iterj} \leftarrow s/\text{chunksize}[2] \quad /\!*\text{ number of chunks for the loop } j \quad */$
3. for $\text{ci} \leftarrow 0 \text{ to } \text{iteri} - 1$
   4. for $\text{cj} \leftarrow 0 \text{ to } \text{iterj} - 1$
      5. for $\text{bi} \leftarrow 1 \text{ to } \text{chunksize}[1]$
         6. for $\text{bj} \leftarrow 1 \text{ to } \text{chunksize}[2]$
            7. $i \leftarrow \text{ci} \times \text{iteri} + \text{bi}$
            8. $j \leftarrow \text{cj} \times \text{iterj} + \text{bj}$
            9. $l \leftarrow Taddr_{\delta,\delta'}[i, j]$  
            10. $BH_{\delta+\delta'}(c)[l] \leftarrow BH_{\delta+\delta'}(c)[l] + BH_\delta(a)[i] \times BH_{\delta'}(b)[j]$
      end
   end
4. /* if $s$ not divisible by chunksize[2] */
5. for $\text{bi} \leftarrow 1 \text{ to } \text{chunksize}[1]$
   6. for $\text{j} \leftarrow \text{iterj} \times \text{chunksize}[2] \text{ to } s$
      7. $i \leftarrow \text{ci} \times \text{iteri} + \text{bi}$
      8. $l \leftarrow Taddr_{\delta,\delta'}[i, j]$  
      9. $BH_{\delta+\delta'}(c)[l] \leftarrow BH_{\delta+\delta'}(c)[l] + BH_\delta(a)[i] \times BH_{\delta'}(b)[j]$
   end
7. end

Function $\text{fmafull}(BH_\delta(a), BH_{\delta'}(b), Taddr_{\delta,\delta'}, \text{chunksize}, BH_{\delta+\delta'}(c))$

Compute the full fused multiplication-addition using the addressing table and cache blocking technique
Benchmark of the cache blocking technique

Factor of the reduction of the execution time for the product of two homogeneous blocks in 8 variables of degree 7

G5 processor

Core2 duo processor
Benchmark of the cache blocking technique

- Factor of the reduction of the execution time of the product of two homogeneous blocks in 8 variables of degree 9
### Benchmarks

\[ s \times (s + 1) \text{ with } s = (1 + x + y + z + t + u)^{14} \]

<table>
<thead>
<tr>
<th>CAS</th>
<th>representation</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ginac 1.3.2</td>
<td>tree</td>
<td>4137</td>
</tr>
<tr>
<td>Maple 10</td>
<td>DAG</td>
<td>2899.70</td>
</tr>
<tr>
<td>Singular 3.0.2</td>
<td>list</td>
<td>144.27</td>
</tr>
<tr>
<td>Maxima 5.9.2</td>
<td>recursive list</td>
<td>443.95</td>
</tr>
<tr>
<td>Mathematica 5.2</td>
<td>tree</td>
<td>766.65</td>
</tr>
<tr>
<td>TRIP 0.99</td>
<td>recursive vector</td>
<td>13.50</td>
</tr>
<tr>
<td>TRIP 0.99</td>
<td>recursive list</td>
<td>12.85</td>
</tr>
<tr>
<td>TRIP 0.99</td>
<td>flat vector</td>
<td>28.10</td>
</tr>
<tr>
<td>TRIP 0.99</td>
<td>homogeneous blocks (with initialization)</td>
<td>5.44</td>
</tr>
<tr>
<td></td>
<td>(after initialization)</td>
<td>0.57</td>
</tr>
</tbody>
</table>
Effect of the sparsity of the polynomials

s × s with s containing some terms of \((1 + x_1 + x_2 + x_3 + x_4 + x_5)^{10}\)
The serie $V$ has 3052 terms and the result has 227453 terms

$$V(\lambda, \lambda', X, X, Y, Y, X', X', Y, Y') = \sum X^{d_1} X^{d_2} Y^{d_3} Y^{d_4} X^{d_5} X^{d_6} Y^{d_7} Y^{d_8} e^{(k_1 \lambda + k_2 \lambda')}$$

$$d_1 + d_2 + d_3 + d_4 + d_5 + d_6 + d_7 + d_8 = 7$$

<table>
<thead>
<tr>
<th>CAS</th>
<th>representation</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maple 10</td>
<td>DAG</td>
<td>345.08</td>
</tr>
<tr>
<td>Mathematica 5.2</td>
<td>tree</td>
<td>149.63</td>
</tr>
<tr>
<td>TRIP 0.99</td>
<td>recursive vector</td>
<td>2.91</td>
</tr>
<tr>
<td>TRIP 0.99</td>
<td>recursive list</td>
<td>2.32</td>
</tr>
<tr>
<td>TRIP 0.99</td>
<td>flat vector</td>
<td>1.97</td>
</tr>
<tr>
<td>TRIP 0.99</td>
<td>homogeneous blocks (with initialization)</td>
<td>2468.10</td>
</tr>
<tr>
<td></td>
<td>homogeneous blocks (after initialization)</td>
<td>2403.45</td>
</tr>
<tr>
<td>TRIP 0.99</td>
<td>compacted homogeneous blocks (with initialization)</td>
<td>17.07</td>
</tr>
<tr>
<td></td>
<td>compacted homogeneous blocks (after initialization)</td>
<td>15.11</td>
</tr>
<tr>
<td>TRIP 0.99</td>
<td>d’alembert blocks (with initialization)</td>
<td>22.55</td>
</tr>
<tr>
<td></td>
<td>d’alembert blocks (after initialization)</td>
<td>8.55</td>
</tr>
<tr>
<td>TRIP 0.99</td>
<td>compacted d’alembert blocks (with initialization)</td>
<td>11.01</td>
</tr>
<tr>
<td></td>
<td>compacted d’alembert blocks (after initialization)</td>
<td>1.25</td>
</tr>
</tbody>
</table>
full product $V \times V$ for different degrees

- recursive vector
- recursive list
- flat vector
- BH
- BHC
- BHDAL
- BHDALC

Graph showing the time (s) for different degrees.
Karatsuba's algorithm

Idea: one multiplication could be avoided for polynomial of degree 1

Let $A$ and $B$ polynomials

$$A(X) = a_0 + a_1X \text{ and } B(X) = b_0 + b_1X$$

The naive multiplication $C = AB$ is

$$C(X) = a_0b_0 + (a_0b_1 + a_1b_0)X^1 + a_1b_1X^2$$

But the coefficient of $X^1$ could be written as

$$a_0b_1 + a_1b_0 = (a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1$$

we need to perform 3 multiplications and 4 additions

instead of 4 multiplications and 1 addition.

could be applied recursively to polynomials of degree $2^{k-1}$

complexity $O(n^{1.59})$
Algorithm 11: Compute the full product of two polynomials $A$ and $B$ using the Karatsuba’s multiplication algorithm

Input: $A$: polynomial of degree at most $n - 1$ with $n = 2^k$ for $k \in \mathbb{N}$
Input: $B$: polynomial of degree at most $n - 1$
Output: $C$: polynomial

if $n = 1$ then return $C \leftarrow AB$
$C_1 \leftarrow A^{(0)}B^{(0)}$ by a recursive call
$C_2 \leftarrow A^{(1)}B^{(1)}$ by a recursive call
$C_3 \leftarrow A^{(0)} + A^{(1)}$
$C_4 \leftarrow B^{(0)} + B^{(1)}$
$C_5 \leftarrow C_3C_4$ by a recursive call
$C_6 \leftarrow C_5 - C_1 - C_2$
$C \leftarrow C_1 + C_6X^{n/2} + C_2X^n$
return $C$
Truncated product

**Univariate polynomials**

\[(a_0 + a_1 X + \ldots + a_n X^n + O(X^n)) \otimes (b_0 + b_1 X + \ldots + b_n X^n + O(X^n)) = (c_0 + c_1 X + \ldots + c_n X^n + O(X^n))\]

**Multivariate polynomials**

- keep the term \(a_i X_1^{d_1} X_2^{d_2} \ldots X_n^{d_n} \otimes b_j X_1^{d'_1} X_2^{d'_2} \ldots X_n^{d'_n}\)

  if \(d_1 + d'_1 + d_2 + d'_2 + \ldots + d_n + d'_n \leq T\)

- if truncation is performed only on some variables, truncated variables must be ordered
Function fmatruncated(A,B,C, T) Compute the truncated fused multiplication-addition 
C = C + A × B with A, B and C multivariate polynomials represented as recursive list

Input: A: polynomial { list of ( coefficients a , degree δ_a) }
Input: B: polynomial { list of ( coefficients b , degree δ_b) }
Input: C: polynomial { list of ( coefficients c , degree δ_c)}
Input: T: degree of the truncation
Output: C: polynomial { list of ( coefficients c , degree δ_c)}

if variable of A is truncated then
    iter ← head of C
    foreach element in A such that δ_a ≤ T do
        /* avoid to scan to C when the loop on B is finished */
        iterb ← iter
        foreach element in B such that δ_a + δ_b ≤ T do
            /* find after iterb in C if the degree δ_a + δ_b is present */
            while current degree δ_c referenced by iterb < δ_a + δ_b do
                iterb ← next element after iterb
            end
            if δ_c = δ_a + δ_b then
                fmatruncated (a, b, c, T - δ_a + δ_b)
                if c = 0 then remove the element referenced by iterb
            else
                insert an element (multruncated (a, b, T - δ_a + δ_b), δ_a + δ_b) just before iterb
            end
        end
        if current element is the first element of B then
            iter ← iterb
        end
    end
else
    fmafull (A,B,C)
end
Truncated product on homogeneous blocks

Let

\[ A = \sum BH_\delta(a), \quad B = \sum BH_\delta(b) \]

Truncated product on the total degree

\[ C = A \bigotimes B = \sum_{\delta=0}^{T} BH_\delta(c) \text{ with } BH_\delta(c) = \sum_{n=0}^{\delta} BH_n(a) \times BH_{\delta-n}(b) \]

- use the full product of 2 homogeneous blocks
- use less addressing tables than for the full product
Truncated product - benchmarks

- recursive vector
- recursive list
- flat vector
- BH
- BHC
- BHDAL
- BHDALC

Graph showing performance metrics for different algorithms as a function of degree and time.
Perform the product of two Poisson series but we only want to keep terms which have specific values for $k_1$ and $k_2$

e.g., $S_1 \times S_2 = S$, we want only terms such that $k_1 = 0$ and $k_2 = 0$

$$S = \sum a_i X_1^{d_1} X_2^{d_2} \ldots X_n^{d_n} \exp(\imath (k_1 \lambda + k_2 \lambda'))$$

very easy for the recursive representation if the series are correctly ordered.
Special truncated product on magnitude

\[ a_i X_1^{d_1} X_2^{d_2} \ldots X_n^{d_n} \otimes b_j X_1^{d'_1} X_2^{d'_2} \ldots X_n^{d'_n} \text{ is kept if } |a_i b_j| \geq \epsilon_0 \]

brut-force method

**Algorithm 1**: Compute the truncated product of the series \( A \) and \( B \) in the amplitude of their coefficients

**Input**: \( A \): serie \( \sum a_i x^i \) ordered by decreasing amplitude  
**Input**: \( B \): serie \( \sum b_i x^i \) ordered by decreasing amplitude  
**Input**: \( \epsilon_0 \): threshold > 0  
**Output**: \( C \): series \( C = AB \) with all coefficients greater than \( \epsilon_0 \)

\( C \leftarrow \) create an empty polynomial

\[
\text{foreach coefficient } a_i \text{ such that } |a_i b_0| \geq \epsilon_0 \text{ do}
\]

\[
\quad \text{foreach coefficient } b_j \text{ such that } |b_j| \geq \epsilon_0 / |a_i| \text{ do}
\]

\[
\quad \quad C \leftarrow C + a_i b_j x^{i+j}
\]

end

end

return \( C \)

order the series on the magnitude of the coefficient?
Special truncated product on magnitude

Let a variable $\epsilon$, and a small parameter $\epsilon'_0$ such that $\epsilon'^p_0 = \epsilon_0$ with $p \in \mathbb{N}$. Each coefficient $a_i$ of the serie $A(x)$ could be written as

$$a_i = a'_i x^i \epsilon^k \text{ with } k = \left\lfloor \frac{\log |a_i|}{\log \epsilon'_0} \right\rfloor \text{ and } a'_i = \frac{a_i}{\epsilon'_0^k}$$

$$A'(\epsilon, x) = \sum_k \left( \sum_j a'_j x^j \right) \epsilon^k$$

$$A(x) = A'(\epsilon'_0, x)$$

The truncated product $A(x) \otimes B(x)$ on the amplitude of the coefficient is transformed to a truncated product $A'(\epsilon, x) \otimes B'(\epsilon, x)$ on the variable $\epsilon$. The degree of truncation is $p$. 
source code

```c
s1 = (1 + 0.05*x) ** 3;
s2 = (1 + 0.04*x) ** 4;
/* introduce the variable eps in s1 and s2 */
s1e = sereps(s1, eps, 0.1);
s2e = sereps(s2, eps, 0.1);
/* define the truncature on amplitude to (0.1)^2 */
tr = {eps, 2};
usetronc(tr);
/* perform the product */
s3e = s1e * s2e;
/* remove the variable eps from s3e */
s3 = invsereps(s3e, eps, 0.1);
```

Execution of the previous source code by trip

\[
\begin{align*}
s1(x) &= 1 + 0.15x + 0.0075x^2 + 0.000125x^3 \\
s2(x) &= 1 + 0.16x + 0.0096x^2 + 0.000256x^3 + 2.56E-06x^4 \\
s1e(x, eps) &= 1 + 0.15x + 0.75x^2*eps + 0.125x^3*eps^2 \\
s2e(x, eps) &= 1 + 0.16x + 0.96x^2*eps + 0.256x^3*eps^2 + 0.256x^4*eps^3 \\
tr &= ( \{ \text{eps}, 2 \} ) \\
s3e(x, eps) &= 1 + 0.31x + 0.024x^2 + 1.71x^2*eps + 0.264x^3*eps^2 \\
s3(x) &= 1 + 0.31x + 0.0411x^2 + 0.00264x^3
\end{align*}
\]