Multiplication of polynomials

M. Gastineau

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

gastineau@imcce.fr

Institut de Mécanique Céleste et de Calcul des Éphémérides

CENTRE NATIONAL
DE LA RECHERCHE
SCIENTIFIQUE

l'Observatoire de Paris

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

$$A(x) = \sum_{i=da_{min}}^{da_{max}} a_i x^i \text{ and } B(x) = \sum_{i=db_{min}}^{db_{max}} b_i x^i$$

- Perform the multiplication of all terms

$$C(x) = \sum_{k=da_{min}+db_{min}}^{da_{max}+db_{max}} c_k x^k \text{ with } c_k = \sum_{i+j=k} a_i b_j$$

- if *A*, *B* and *C*, have *r*, *s* and *t* terms
  - *rs* multiplications and *rs*-*t* additions
  - complexity : *O(rs)*

Multiplication for univariate polynomials stored as vector

Algorithm 1: Compute the full product of univariate polynomials A and B represented with a dense vector

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

Multiplication for recursive dense vector 1/2

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

Input: A: multivariate polynomial $\{da_{min}, da_{max}$, array of coefficients $a\}$
Input: B: multivariate polynomial $\{db_{min}, db_{max}$, array of coefficients $b\}$
Output: C: multivariate polynomial $\{dc_{min}, dc_{max}$, 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**
    $c[k] \leftarrow$ `mulfull` $(a[da_{min}], b[k - da_{min}])$
    for $j \leftarrow da_{min} + 1$ **to** da_{max} **do**
        `fmafull` $(a[j], b[k - j], c[k])$
    end
end
return C

Multiplication for recursive dense vector 2/2

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 dense vector

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

$newdc_{min} \leftarrow da_{min} + db_{min}$
$newdc_{max} \leftarrow da_{max} + db_{max}$
if $newdc_{min} < dc_{min}$ *or* $dc_{max} < newdc_{max}$ **then**
    $dc_{min} \leftarrow \min(newdc_{min}, dc_{min})$
    $dc_{max} \leftarrow \max(newdc_{max}, dc_{max})$
    resize C
end
for $k \leftarrow dc_{min}$ **to** dc_{max} **do**
    $c[k] \leftarrow$ `mulfull` $(a[da_{min}], b[k - da_{min}])$
    for $j \leftarrow da_{min} + 1$ **to** da_{max} **do**
        `fmafull` $(a[j], b[k-j], c[k])$
    end
end
if *c contains 0 at its beginning or at its end* **then**
    adjust dc_{min}
    adjust dc_{max}
    resize C
end

Multiplication for univariate polynomials stored as list

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

$C \leftarrow$ create a empty polynomial
foreach *element in* A **do**
  $D \leftarrow$ create a empty polynomial
  foreach *element in* B **do**
    add to the tail of D an element $(a \times b,\ \delta_a + \delta_b)$
  end
  $C \leftarrow C + D$
end
return C

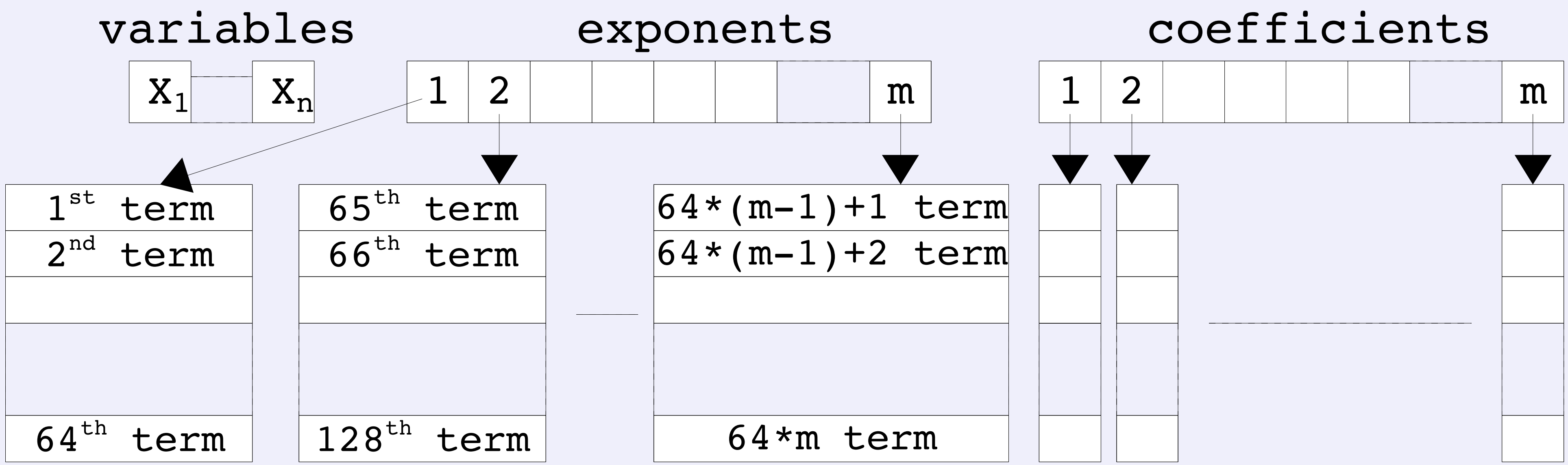
Multiplication for recursive list

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

iter ← head of C
foreach *element in* A **do**
  `// avoid to scan to` C `when the loop on` B `is finished`
  iterb ← *iter*
  foreach *element in* B **do**
    `// find after` *iterb* `in` C `if the degree` $\delta_a + \delta_b$ `is present`
    while *current degree* δ_c *referenced by iterb* $< \delta_a + \delta_b$ **do**
      iterb ← next element after *iterb*
    end
    if $\delta_c = \delta_a + \delta_b$ **then**
      `fmafull` (a, b, c)
      if $c = 0$ **then** remove the element referenced by *iterb*
    else
      insert an element (`mulfull` (a, b), $\delta_a + \delta_b$) just before *iterb*
    end
    if *current element is the first element of* B **then**
      iter ← *iterb*
    end
  end
end

Multiplication for flat vector 1/2

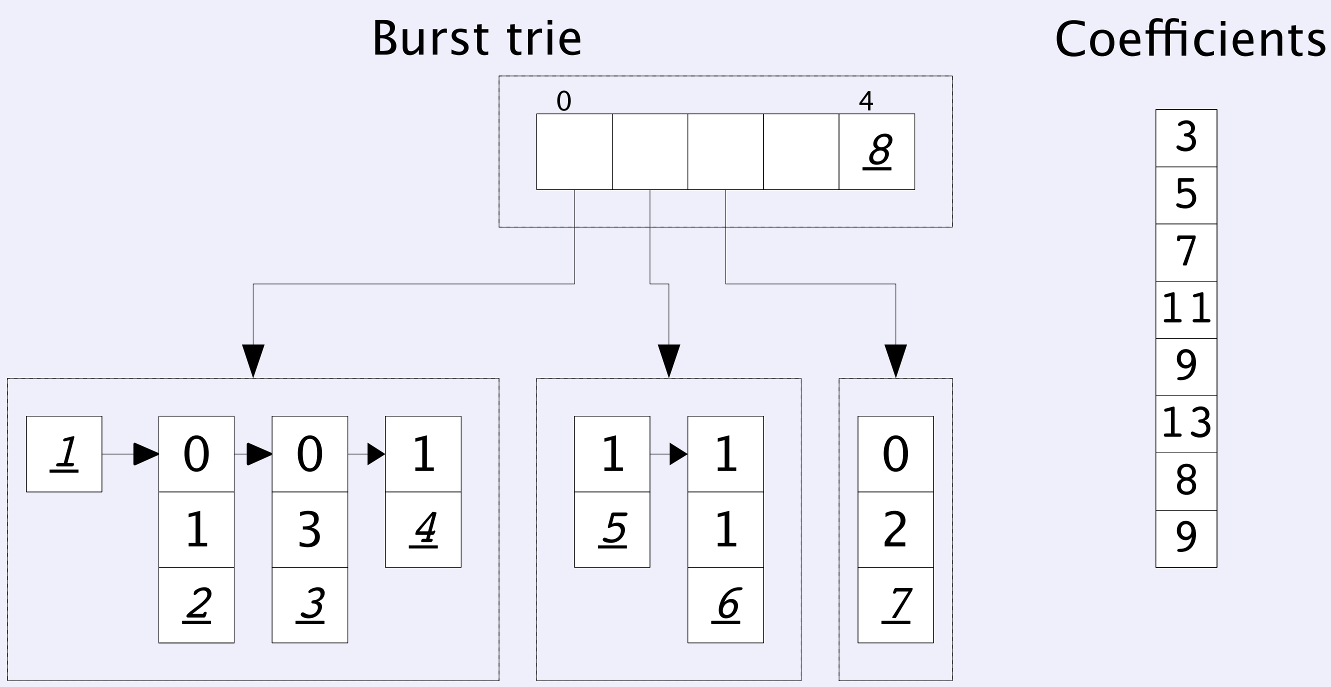


- How to sort terms ?
  - search and shift operations too slow
  - need an intermediate and adjustable storage : BURST TRIE

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

$$A = \sum_{\delta=da_{min}}^{da_{max}} BH_\delta(a) \quad , \quad B = \sum_{\delta=db_{min}}^{db_{max}} BH_\delta(b)$$

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

$$C = A \times B = \sum_{\delta=dc_{min}}^{dc_{max}} BH_\delta(c)$$

with

$$dc_{min} = da_{min} + db_{min}$$

$$dc_{max} = da_{max} + db_{max}$$

$$BH_\delta(c) = \sum_{i+j=\delta} BH_i(a) \times BH_j(b)$$

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 δ, a : array of r coeff. }
Input: $BH_{\delta'}(b)$: homogeneous blocks { degree δ', 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 ← 1 to r do
    for j ← 1 to s do
        l ← get location of the term in BH_{δ+δ'}(c)
        BH_{δ+δ'}(c)[l] ← BH_{δ+δ'}(c)[l] + BH_δ(a)[i] × BH_{δ'}(b)[j]
    end
end
```

Homogeneous block using functions

Function `fmafull`$(BH_\delta(a), BH_{\delta'}(b), BH_{\delta+\delta'}(c))$
Compute the full fused multiplication-addition
$BH_{\delta+\delta'}(c) = BH_{\delta+\delta'}(c) + BH_\delta(a) \times BH_{\delta'}(b)$ using functions to compute location

Input: $BH_\delta(a)$: homogeneous blocks { degree δ, a : array of r coeff. }
Input: $BH_{\delta'}(b)$: homogeneous blocks { degree δ', 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**
  $expoa \leftarrow$ get array of exponents from the location i in BH_δ
  for $j \leftarrow 1$ **to** s **do**
    $expob \leftarrow$ get array of exponents from the location j in BH'_δ
    $expoc \leftarrow expoa + expob$
    $l \leftarrow$ get location of the term with exponents $expoc$ 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.

$$Texp_1 + Texp_2 = Texp_3$$

$Texp_1$:

0	0	1
0	1	0
1	0	0

$Texp_2$:

0	0	2
0	1	1
0	2	0
1	0	1
1	1	0
2	0	0

$Texp_3$:

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

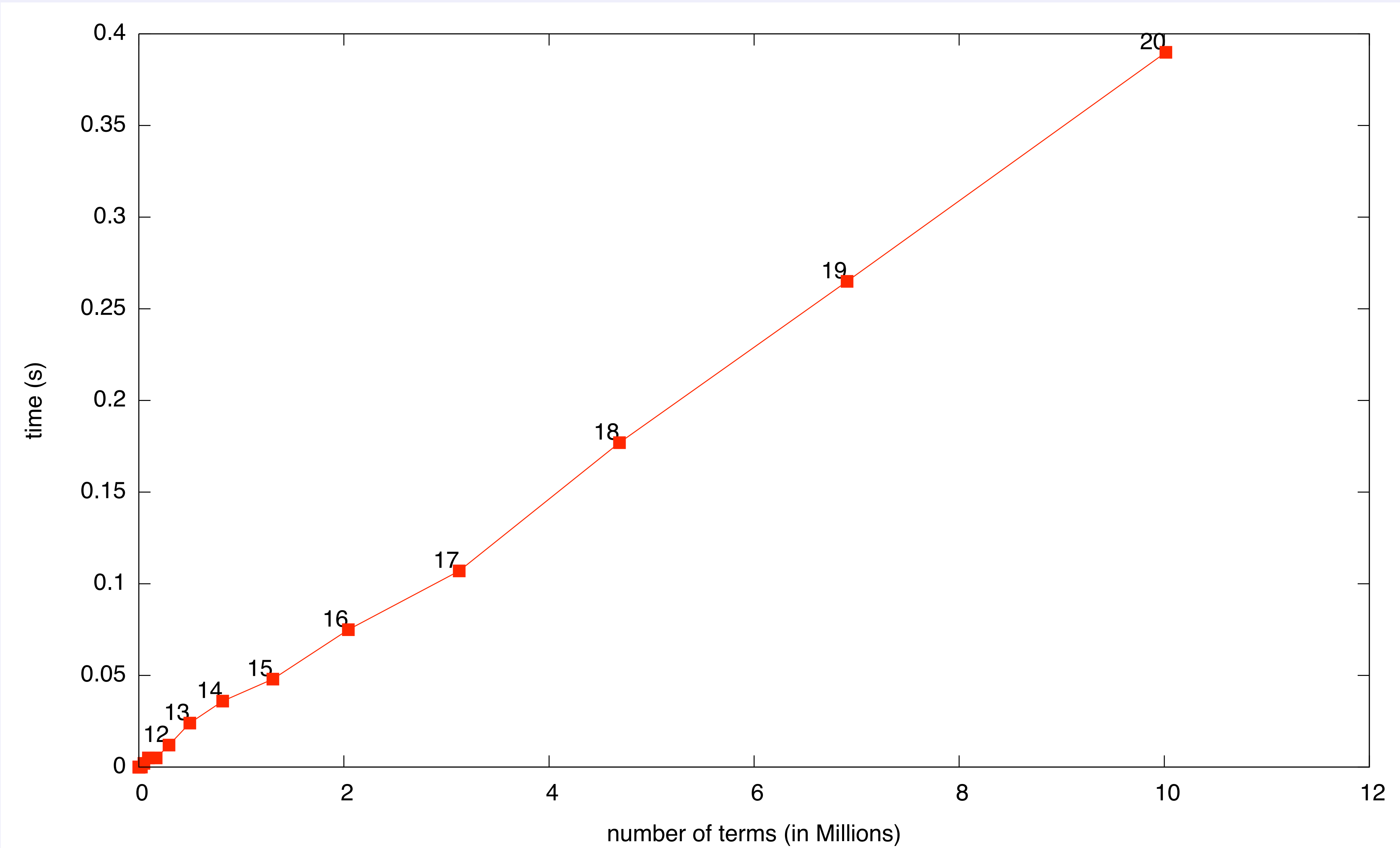
$Taddr_{1,2}$

1	2	3	5	6	8
2	3	4	6	7	9
5	6	7	8	9	10

- $Taddr_{2,1} =^t Taddr_{1,2}$

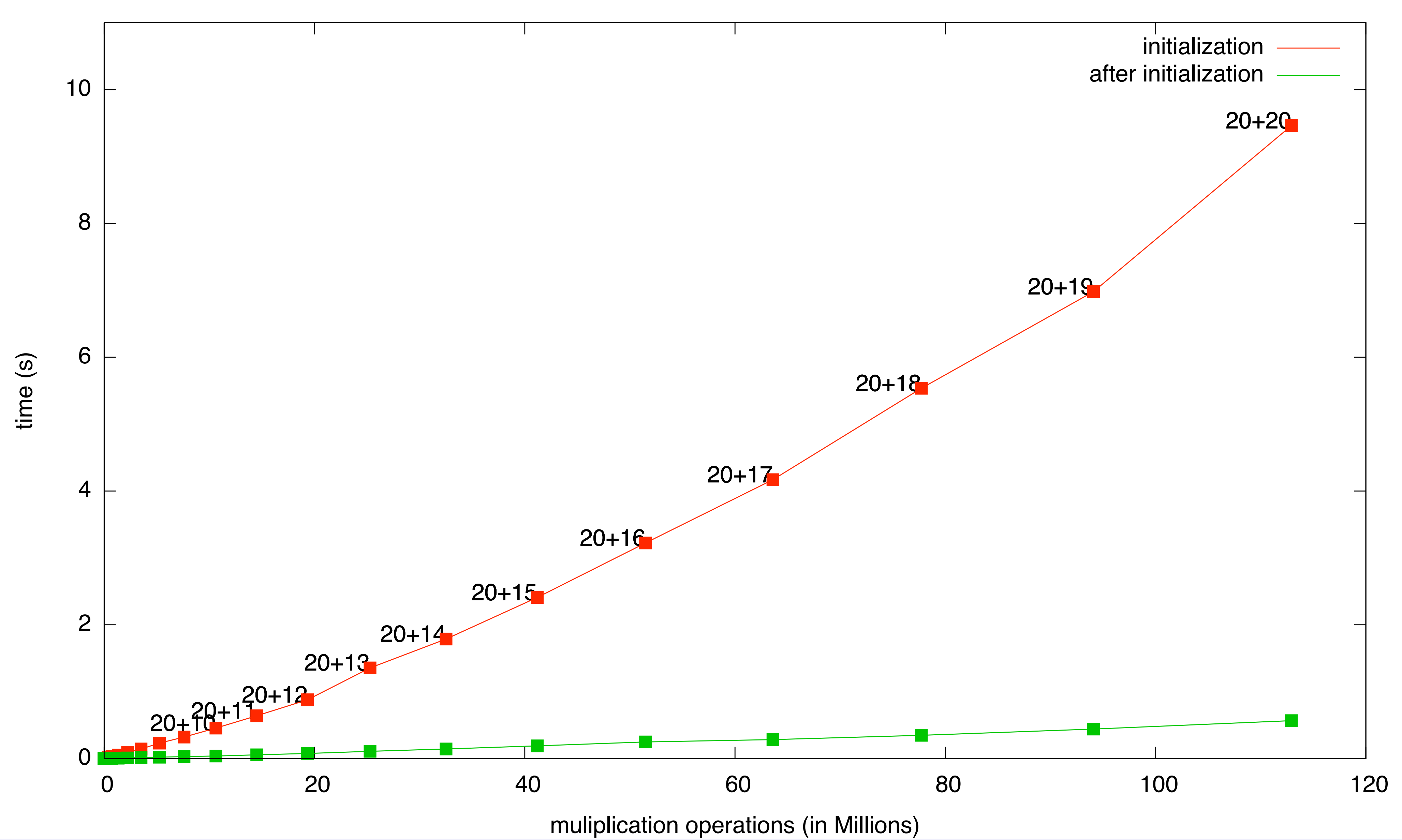
Execution time to build the tables of exponents

- build the tables of exponents for homogeneous blocks in 10 variables up to the degree 20

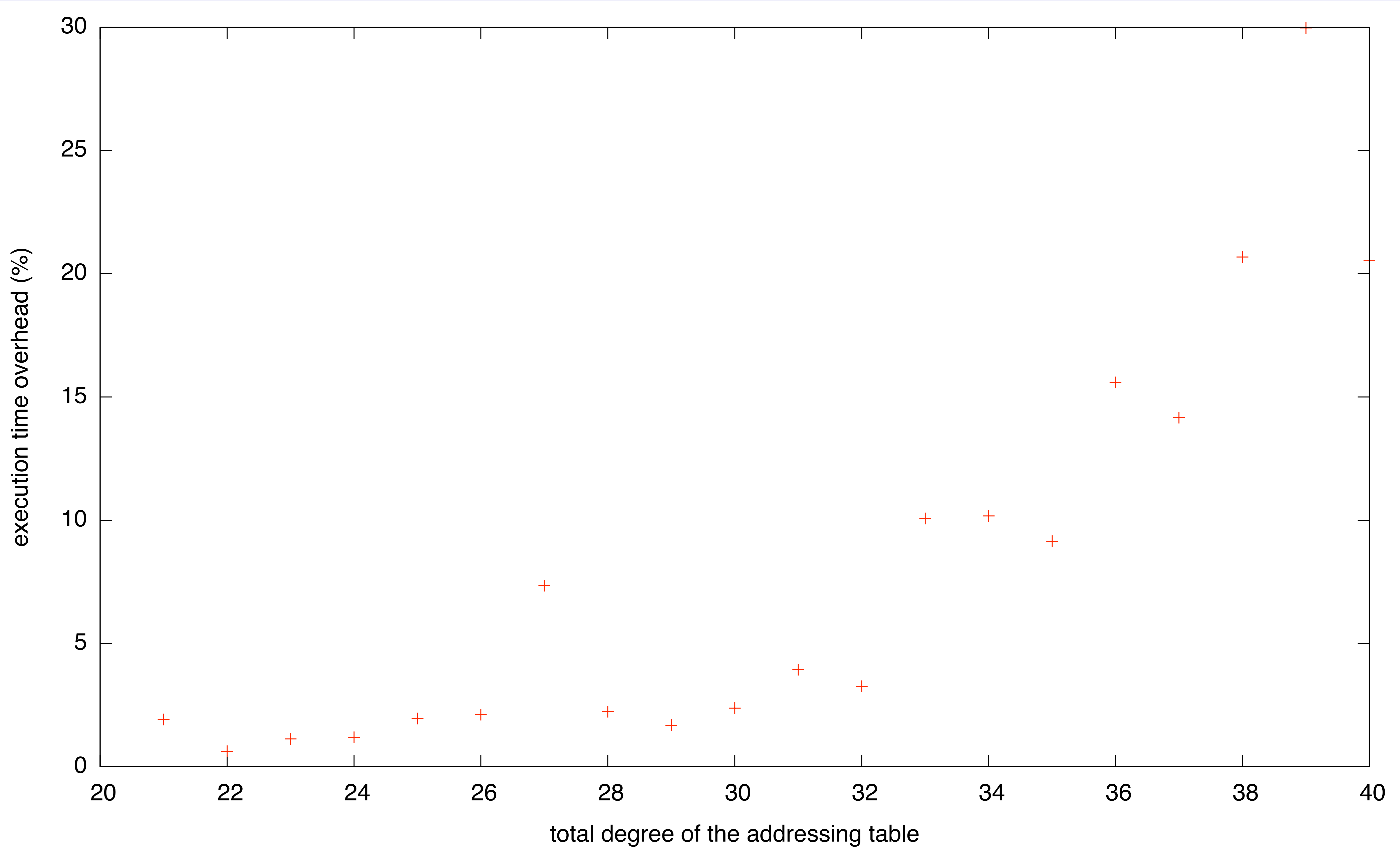


Execution time to build the addressing tables

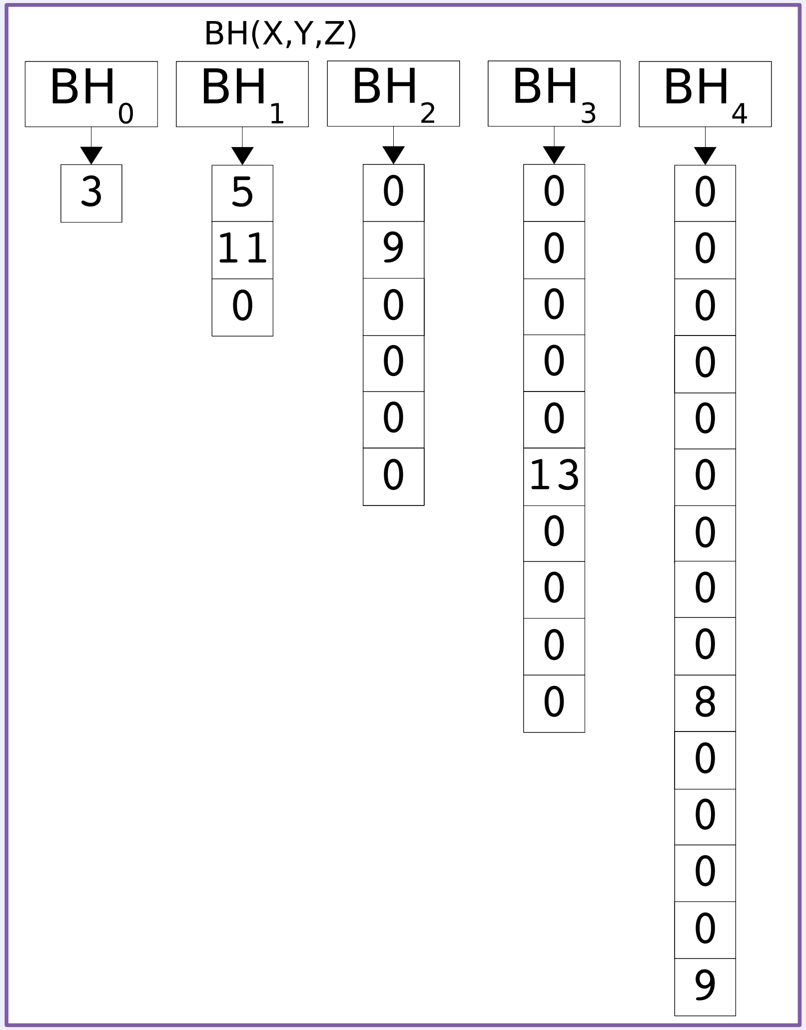
- product of two homogeneous blocks in 5 variables up to the total degree 40



Overhead to load the addressing tables from disk

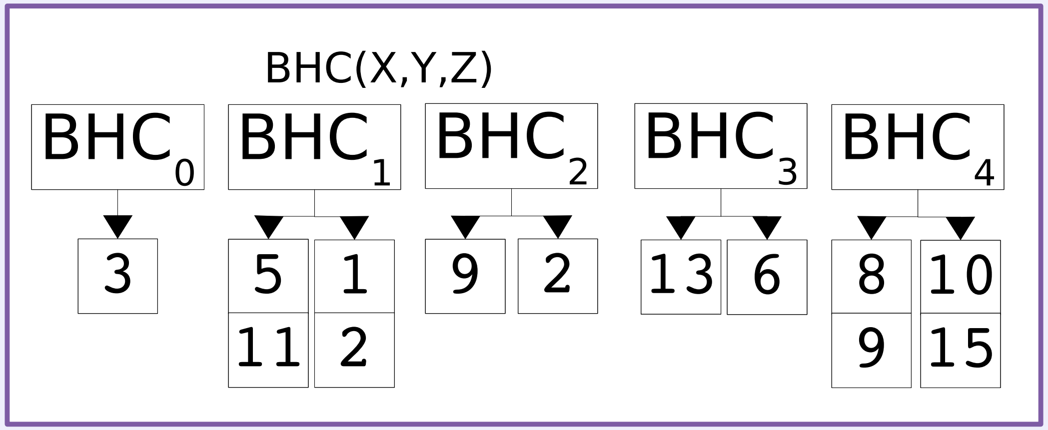


Homogeneous blocks



$$P(x,y,z) = 3 + 5z + 7z^3 + 11y + 9yz + 13xyz + 8x^2z^2 + 9x^4$$

Compacted homogeneous blocks



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 δ, a : array of r coeff. }
Input: $BH_{\delta'}(b)$: homogeneous blocks { degree δ', b : array of s coeff. }
Input: $BH_{\delta+\delta'}(c)$: homog. blocks { degree $\delta+\delta'$, c : array of t coeff. }
Input: $Taddr_{\delta,\delta'}$: addressing table of degree δ, δ'
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**
    $l \leftarrow Taddr_{\delta,\delta'}[i,j]$
    $BH_{\delta+\delta'}(c)[l] \leftarrow BH_{\delta+\delta'}(c)[l] + BH_\delta(a)[i] \times BH_{\delta'}(b)[j]$
  end
end

Homogeneous block using addressing tables

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

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 δ, a : array of r coeff. }

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

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

Input: $Taddr_{\delta,\delta'}$: addressing table of degree δ, δ'

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**
    $l \leftarrow Taddr_{\delta,\delta'}[BHC_\delta(a).index[i], BHC_{\delta'}(b).index[j]]$
    $BH_{\delta+\delta'}(c)[l] \leftarrow BH_{\delta+\delta'}(c)[l] + BH_\delta(a)[i] \times BH_{\delta'}(b)[j]$
  end
end

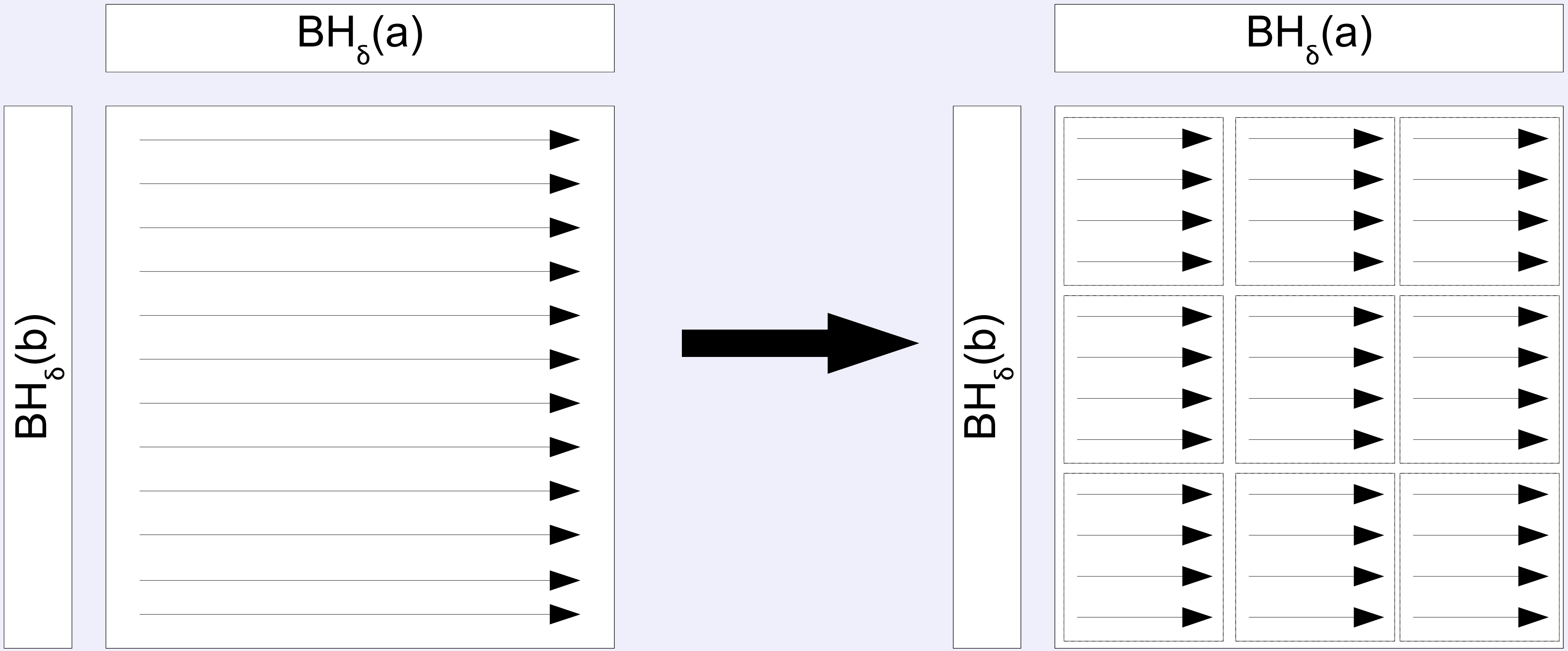
Homogeneous block using addressing tables

Function `fmafull`$(BH_\delta(a),\ BH_{\delta'}(b),\ Taddr_{\delta,\delta'},\ BH_{\delta+\delta'}(c))$
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 δ, a : array of r coeff. }
Input: $BH_{\delta'}(b)$: homogeneous blocks { degree δ', b : array of s coeff. }
Input: $BH_{\delta+\delta'}(c)$: homog. blocks { degree $\delta+\delta'$, c : array of t coeff. }
Input: $Taddr_{\delta,\delta'}$: addressing table of degree δ, δ'
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**
    $l \leftarrow Taddr_{\delta,\delta'}[i,j]$
    $BH_{\delta+\delta'}(c)[l] \leftarrow BH_{\delta+\delta'}(c)[l] + BH_\delta(a)[i] \times BH_{\delta'}(b)[j]$
  end
end

Cache blocking technique



normal flow

flow with cache blocking

Cache blocking technique

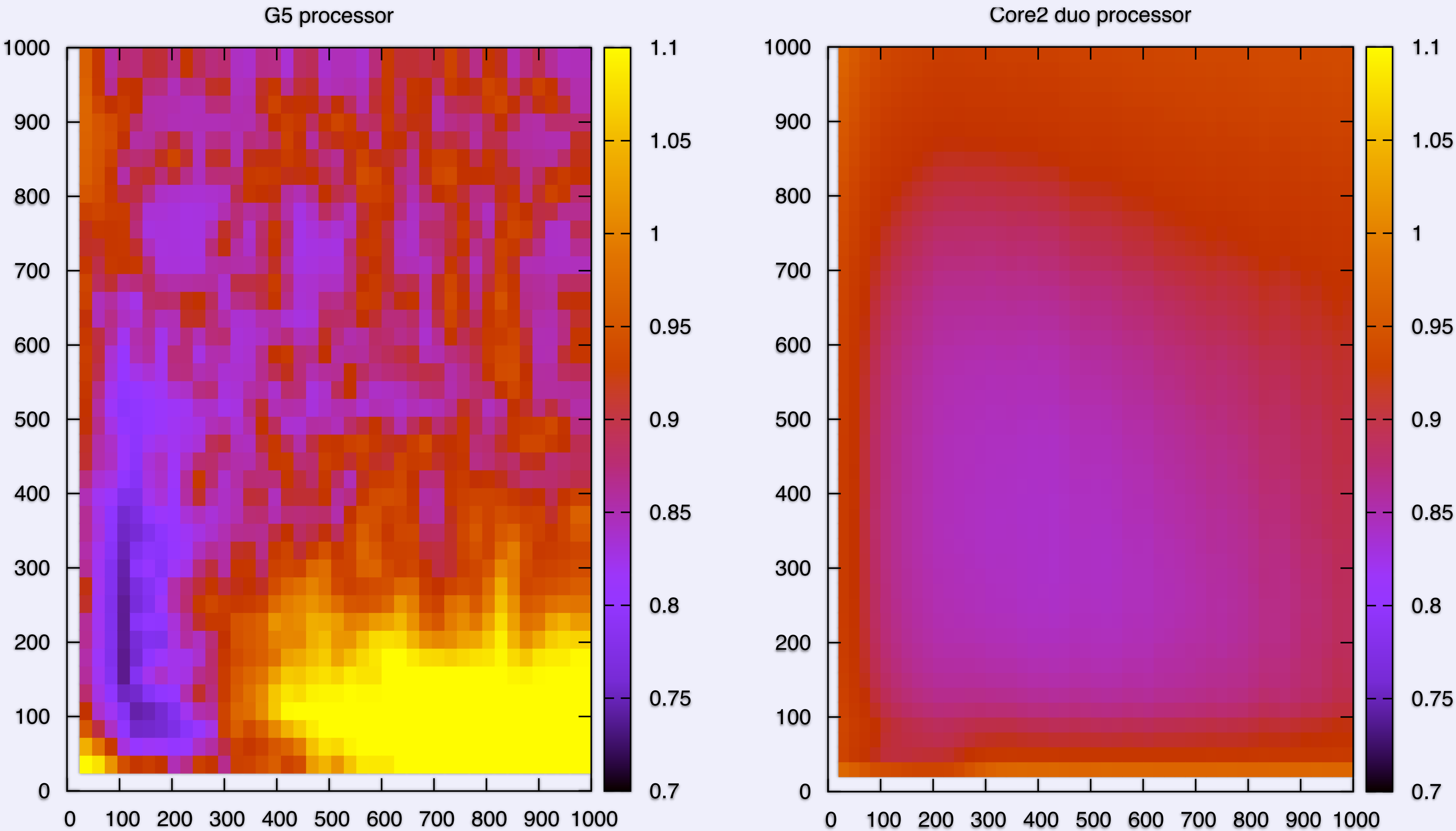
Input: $BH_\delta(a)$: homogeneous blocks { degree δ, a : array of r coeff. }
Input: $BH_{\delta'}(b)$: homogeneous blocks { degree δ', b : array of s coeff. }
Input: $Taddr_{\delta,\delta'}$: addressing table of degree δ, δ'
Input: $BH_{\delta+\delta'}(c)$: homog. blocks { degree $\delta+\delta'$, c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

$iteri \leftarrow r/chunksize[1]$ `/* number of chunks for the loop` i `*/`
$iterj \leftarrow s/chunksize[2]$ `/* number of chunks for the loop` j `*/`
for $ci \leftarrow 0$ **to** $iteri - 1$ **do**
  for $cj \leftarrow 0$ **to** $iterj - 1$ **do**
    for $bi \leftarrow 1$ **to** $chunksize[1]$ **do**
      for $bj \leftarrow 1$ **to** $chunksize[2]$ **do**
        $i \leftarrow ci \times iteri + bi$
        $j \leftarrow cj \times iterj + bj$
        $l \leftarrow Taddr_{\delta,\delta'}[i,j]$
        $BH_{\delta+\delta'}(c)[l] \leftarrow BH_{\delta+\delta'}(c)[l] + BH_\delta(a)[i] \times BH_{\delta'}(b)[j]$
      end
    end
  end
  `/* if` s `not divisible by` $chunksize[2]$ `*/`
  for $bi \leftarrow 1$ **to** $chunksize[1]$ **do**
    for $j \leftarrow iterj \times chunksize[2]$ **to** s **do**
      $i \leftarrow ci \times iteri + bi$
      $l \leftarrow Taddr_{\delta,\delta'}[i,j]$
      $BH_{\delta+\delta'}(c)[l] \leftarrow BH_{\delta+\delta'}(c)[l] + BH_\delta(a)[i] \times BH_{\delta'}(b)[j]$
    end
  end
end
`/* if` r `not divisible by` $chunksize[1]$ `*/`
for $i \leftarrow iteri \times chunksize[1]$ **to** r **do**
  for $j \leftarrow 1$ **to** s **do**
    $l \leftarrow Taddr_{\delta,\delta'}[i,j]$;
    $BH_{\delta+\delta'}(c)[l] \leftarrow BH_{\delta+\delta'}(c)[l] + BH_\delta(a)[i] \times BH_{\delta'}(b)[j]$
  end
end

Function `fmafull`($BH_\delta(a)$, $BH_{\delta'}(b)$, $Taddr_{\delta,\delta'}$, $chunksize$, $BH_{\delta+\delta'}(c)$)
Compute the full fused multiplication-addition using the addressing table and cache blocking technique

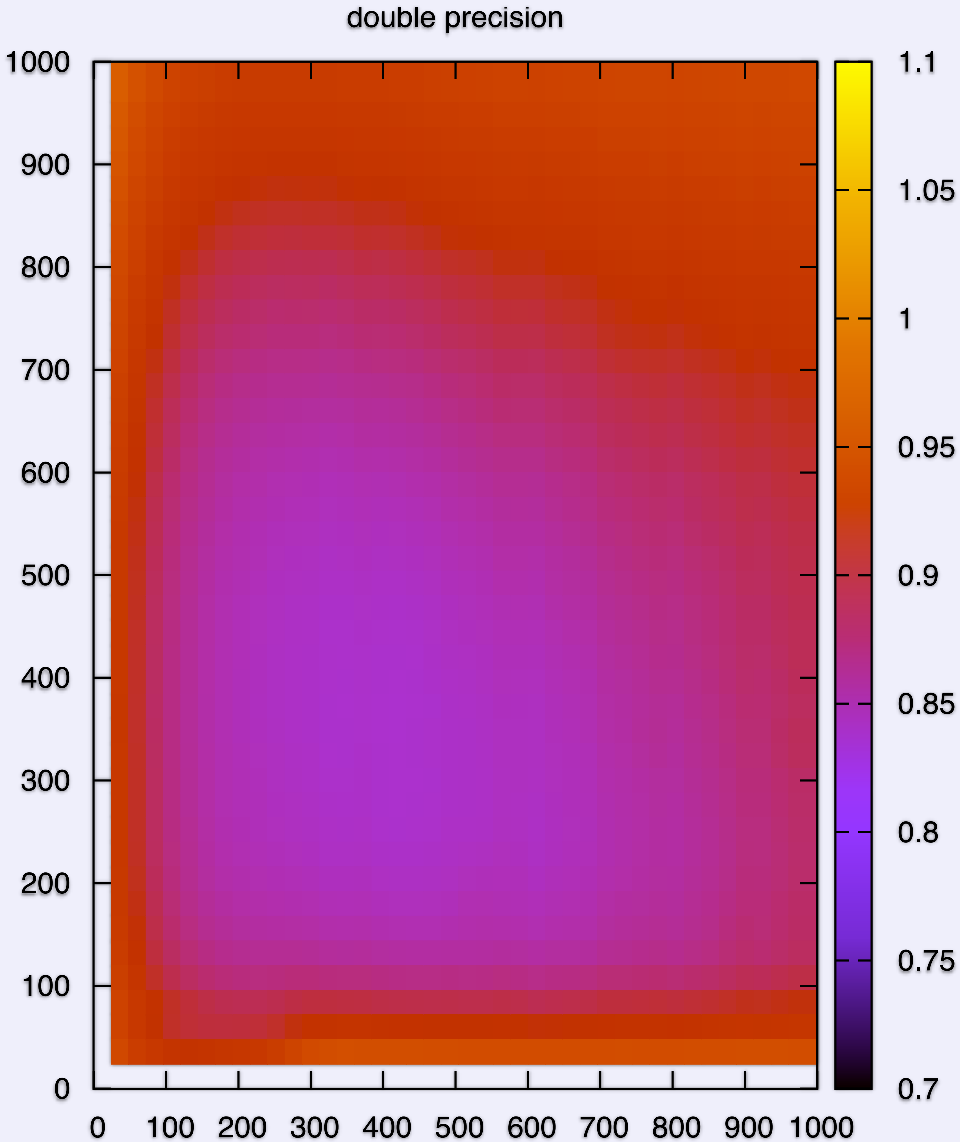
Benchmark of the cache blocking technique

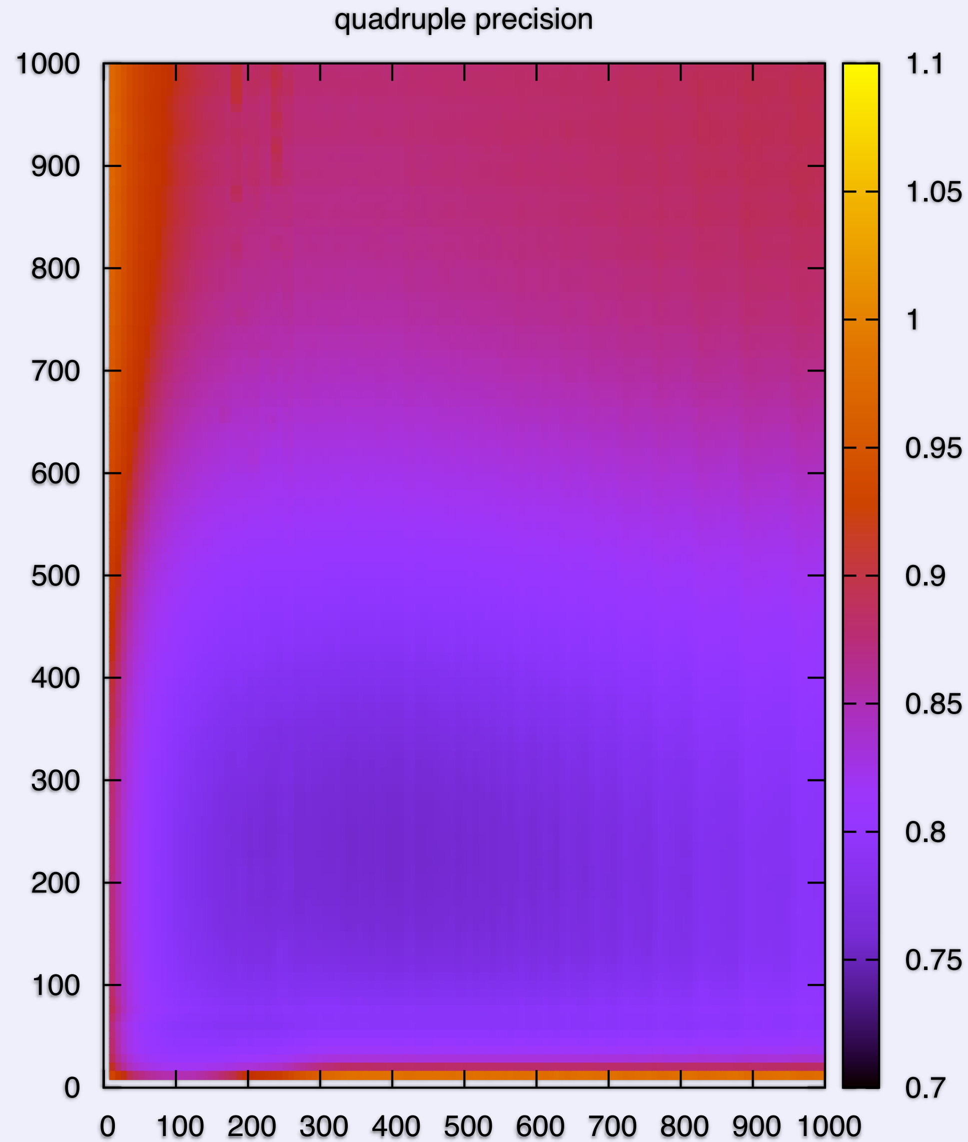
- Factor of the reduction of the execution time for the product of two homogeneous blocks in 8 variables of degree 7



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





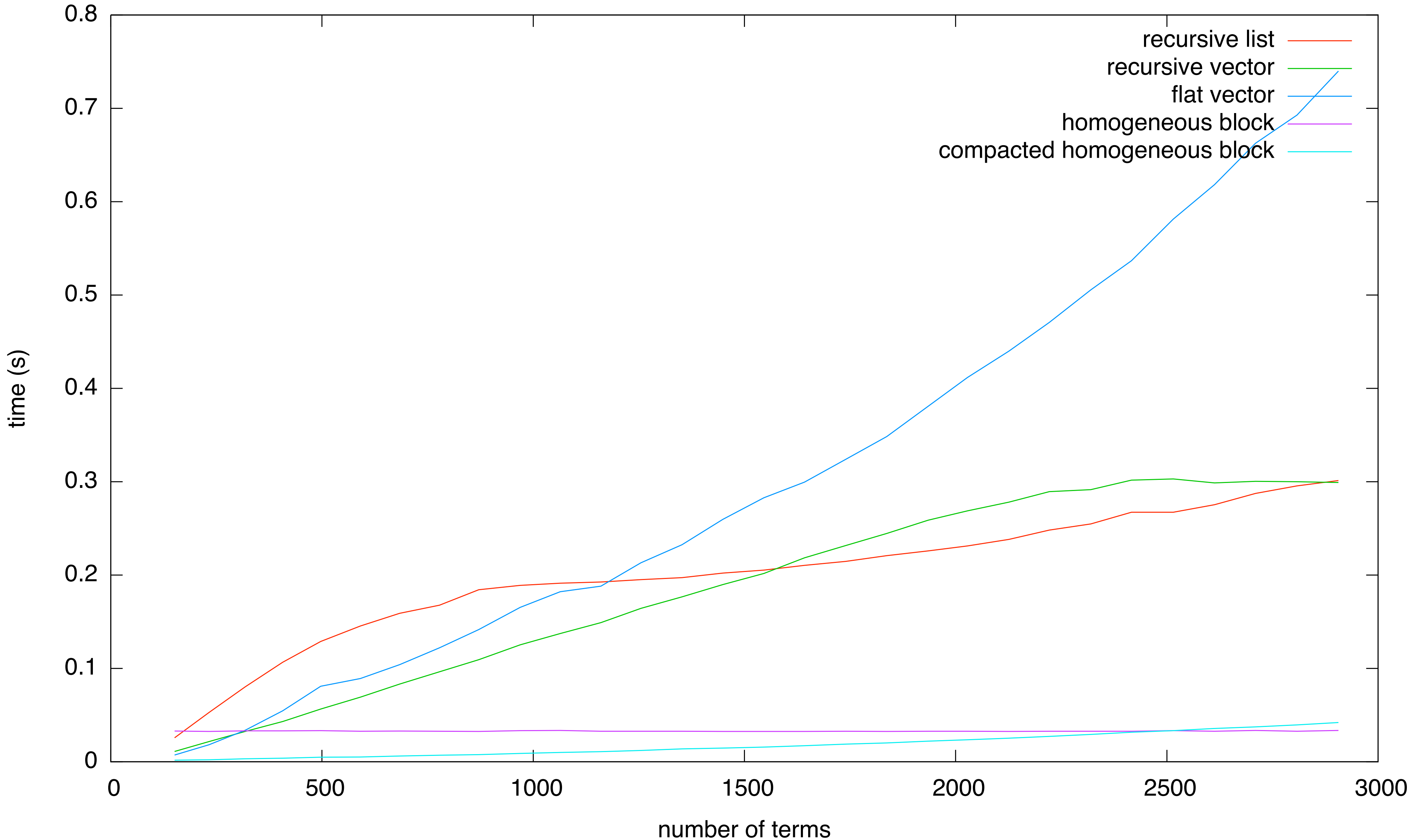
Benchmarks

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

CAS	representation		time (s)
Ginac 1.3.2	tree		4137
Maple 10	DAG		2899.70
Singular 3.0.2	list		144.27
Maxima 5.9.2	recursive list		443.95
Mathematica 5.2	tree		766.65
TRIP 0.99	recursive vector		13.50
TRIP 0.99	recursive list		12.85
TRIP 0.99	flat vector		28.10
TRIP 0.99	homogeneous blocks	(with initialization)	5.44
		(after initialization)	0.57

Effect of the sparsity of the polynomials

- $s \times s$ with s containing some terms of $(1 + x_1 + x_2 + x_3 + x_4 + x_5)^{10}$



full product VxV with different representations

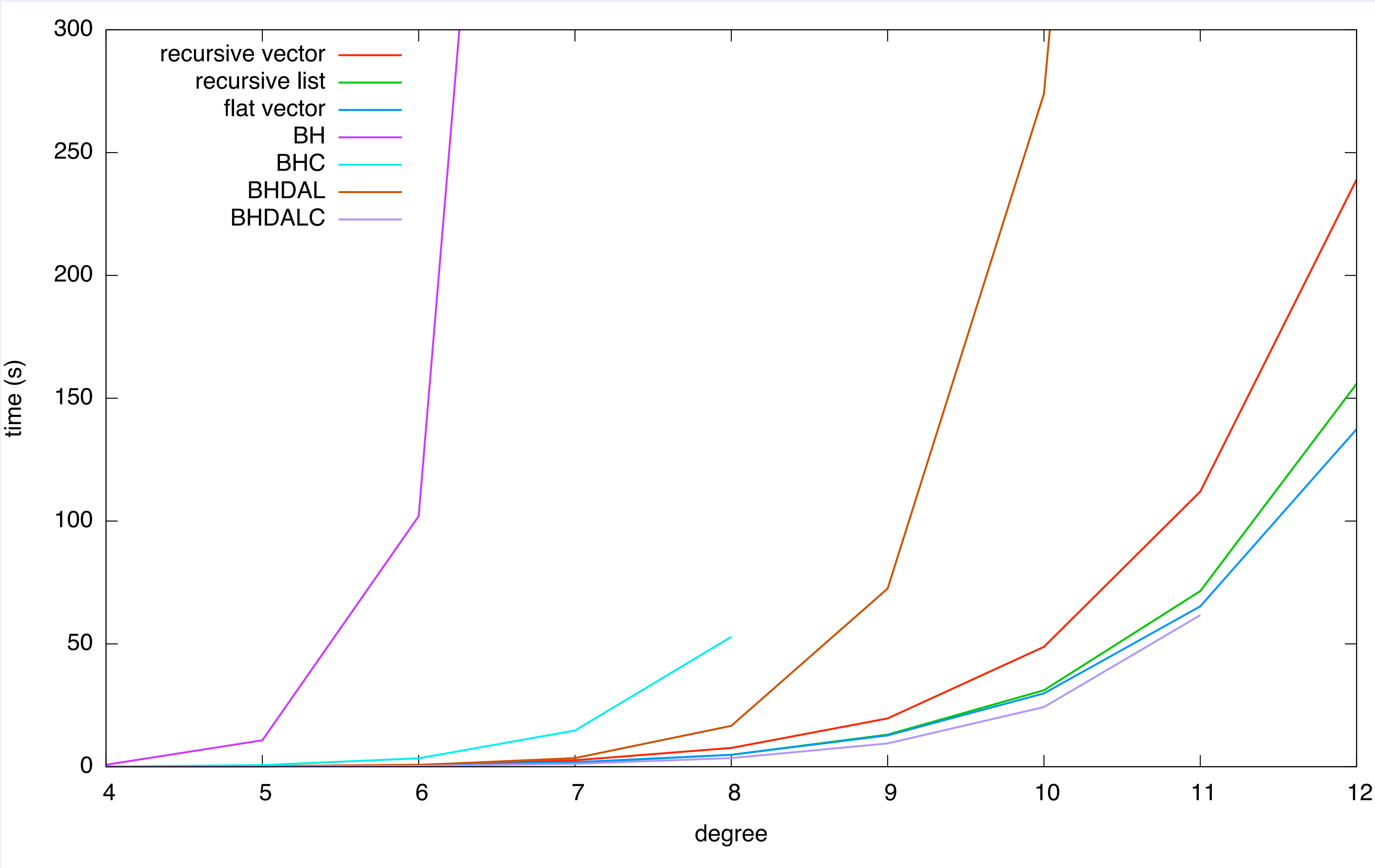
- The serie V has 3052 terms and the result has 227453 terms

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

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

CAS	*representation*		*time (s)*
Maple 10	DAG		345.08
Mathematica 5.2	tree		149.63
TRIP 0.99	recursive vector		2.91
TRIP 0.99	recursive list		2.32
TRIP 0.99	flat vector		1.97
TRIP 0.99	homogeneous blocks	(with initialization) (after initialization)	2468.10 2403.45
TRIP 0.99	compacted homogeneous blocks	(with initialization) (after initialization)	17.07 15.11
TRIP 0.99	d'alembert blocks	(with initialization) (after initialization)	22.55 8.55
TRIP 0.99	compacted d'alembert blocks	(with initialization) (after initialization)	11.01 1.25

full product *VxV* for different degrees



Karatsuba's algorithm

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

  Let A and B polynomials

  $$A(X) = a_0 + a_1 X \text{ and } B(X) = b_0 + b_1 X$$

  The naive multiplication $C = AB$ is

  $$C(X) = a_0 b_0 + (a_0 b_1 + a_1 b_0) X^1 + a_1 b_1 X^2$$

  But the coefficient of X^1 could be written as

  $$a_0 b_1 + a_1 b_0 = (a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1$$

- 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 $O(n^{1.59})$

Karatsuba's algorithm

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

$$(a_0 + a_1 X + ... + a_n X^n + O(X^n)) \bigotimes (b_0 + b_1 X + ... + b_n X^n + O(X^n))$$
$$=$$
$$(c_0 + c_1 X + ... + c_n X^n + O(X^n))$$

- Multivariate polynomials
  - keep the term $a_i X_1^{d_1} X_2^{d_2} ... X_n^{d_n} \bigotimes b_j X_1^{d'_1} X_2^{d'_2} ... X_n^{d'_n}$

    if $d_1 + d'_1 + d_2 + d'_2 + ... + d_n + d'_n \leq T$
  - if truncation is performed only on some variables, truncated variables must be ordered

Truncated product on polynomials stored as recursive list

Function `fmatruncated`(A,B,C, T) Compute the truncated 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 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 $\leftarrow$ head of C
  foreach *element in A such that* $\delta_a \le T$ **do**
    `/* avoid to scan to` C `when the loop on` B `is finished */`
    iterb $\leftarrow$ *iter*
    foreach *element in B such that* $\delta_a + \delta_b \le T$ **do**
      `/* find after` *iterb* `in` C `if the degree` $\delta_a + \delta_b$ `is present */`
      while *current degree* δ_c *referenced by iterb* $< \delta_a + \delta_b$ **do**
        iterb $\leftarrow$ next element after *iterb*
      end
      if $\delta_c = \delta_a + \delta_b$ **then**
        `fmatruncated` $(a, b, c, T - \delta_a + \delta_b)$
        if $c = 0$ **then** remove the element referenced by *iterb*
      else
        insert an element (`multruncated` $(a, b, T - \delta_a + \delta_b)$, $\delta_a + \delta_b$) just before *iterb*
      end
      if *current element is the first element of B* **then**
        iter $\leftarrow$ *iterb*
      end
    end
  end
else
  `fmafull` (A,B,C)
end

Truncated product on homogeneous blocks

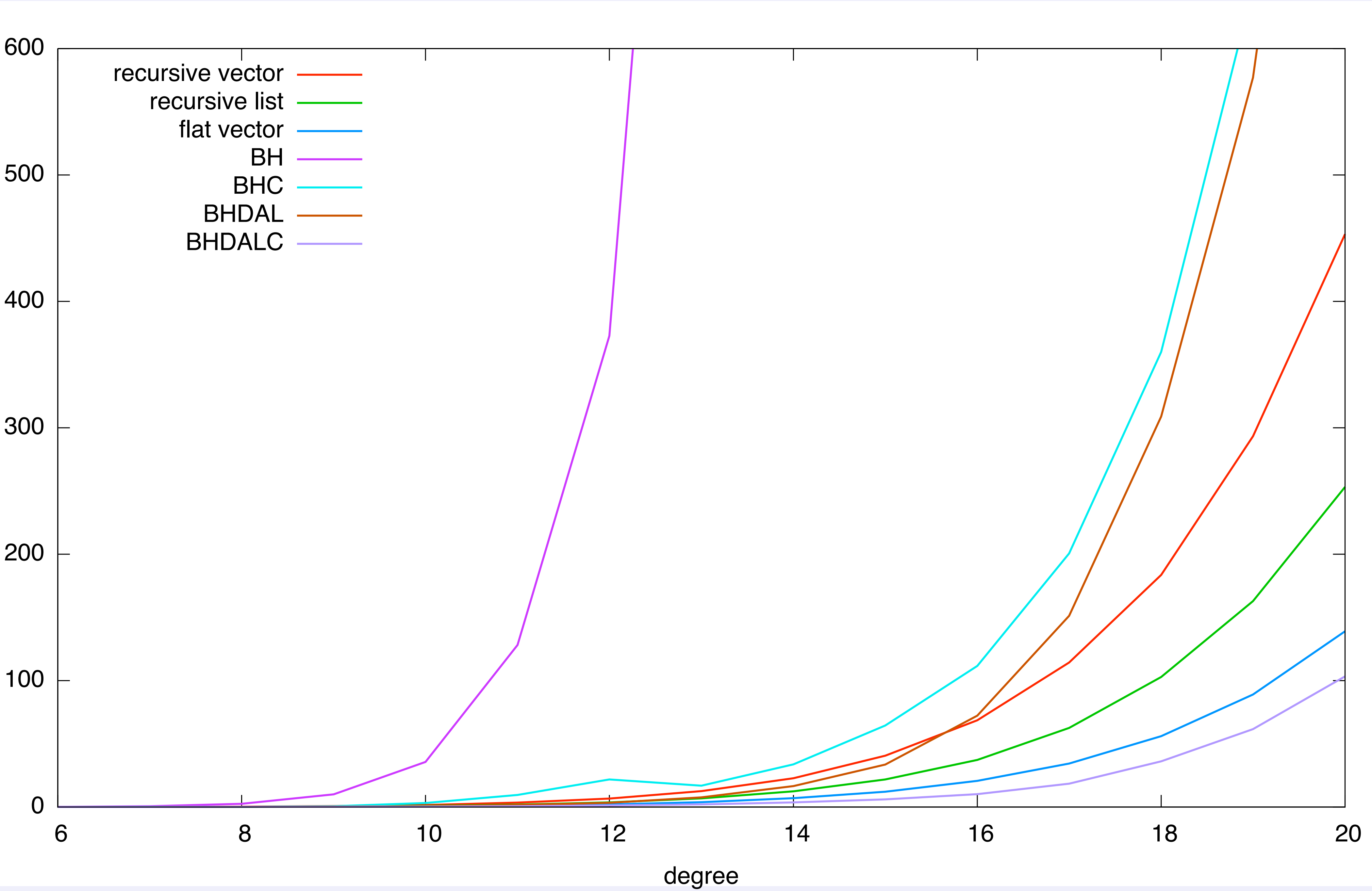
- Let $A = \sum BH_\delta(a) \quad , \quad B = \sum BH_\delta(b)$

- 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



Special truncated product in degree

- 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 $k_2 = 0$

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

- very easy for the recursive representation if the series are correctly ordered.

Special truncated product on magnitude

- $a_i X_1^{d_1} X_2^{d_2} \ldots X_n^{d_n} \otimes b_j X_1^{d'_1} X_2^{d'_2} \ldots X_n^{d'_n}$ 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: ϵ_0: thresold > 0
Output: C: series $C = AB$ with all coefficients greater than ϵ_0

$C \leftarrow$ create an empty polynomial
foreach *coefficient* a_i *such that* $|a_i b_0| \geq \epsilon_0$ **do**
  foreach *coefficient* b_j *such that* $|b_j| \geq \epsilon_0 / |a_i|$ **do**
    $C \leftarrow C + a_i b_j x^{i+j}$
  end
end
return C

- order the series on the magnitude of the coefficient ?

Special truncated product on magnitude

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

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

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

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

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

Special truncated product in amplitude

source code

```
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

```
s1(x) = 1 + 0.15*x + 0.0075*x**2 + 0.000125*x**3
s2(x) = 1 + 0.16*x + 0.0096*x**2 + 0.000256*x**3 + 2.56E-06*x**4
s1e(x,eps) = 1 + 0.15*x + 0.75*x**2*eps**2 + 0.125*x**3*eps**3
s2e(x,eps) = 1 + 0.16*x + 0.96*x**2*eps**2 + 0.256*x**3*eps**3 + 0.256*x**4*eps**5

tr = ( { eps, 2 } )

s3e(x,eps) = 1 + 0.31*x + 0.024*x**2 + 1.71*x**2*eps**2 + 0.264*x**3*eps**2
s3(x) = 1 + 0.31*x + 0.0411*x**2 + 0.00264*x**3
```