# Parallel modular multiplication on multi-core processors

Pascal Giorgi, Laurent Imbert, Thomas Izard[(1)]

CNRS, LIRMM, Université Montpellier 2

(1) Silkan, Montpellier

Arith21, Austin, TX

April 2013

# Motivations

- Fast multiplication is critical for many applications
- Current processors embed several arithmetic units
- Can multiplication (over finite rings) be efficiently parallelized on multi-core processors?
- For which operands' sizes? Using how many cores?

# Multiplication modulo $P$

Euclid: let $C \in \mathbf{Z}$, then $C = PQ + R$, with $R < P$

Modular multiplication: $C = AB \bmod P$ with $A, B < P$

Interleaved multiply-reduce vs multiply then reduce

Notations: We consider multiple precision integers in radix $\beta$

$$C = \sum c_i \beta^i, \quad \text{with } c_i \in \{0, \ldots, \beta - 1\}$$

$M(m, n)$: time to perform an integer multiplication with operands of size $m$ and $n$ respectively, or simply $M(n)$ when both operands have the same size

We shall assume $M(n_1 + n_2, n) = M(n_1, n) + M(n_2, n)$

(although best strategies do exist for unbalanced multiplication. See Brent, Zimmermann's MCA, 2010)

# Barrett's reduction algorithm

Let $0 < P < \beta^n$ and $0 < C < P^2$

($C$ may be the result of a multiplication of $A < P$ and $B < P$)

## Algorithm

1. Compute an approximation of the quotient $\lfloor C/P \rfloor$ as

$$Q = \left\lfloor \frac{\lfloor C/\beta^n \rfloor \times \nu}{\beta^n} \right\rfloor \quad \text{where } \nu = \lfloor \beta^{2n}/P \rfloor \text{ is precomputed}$$

2. Compute $R = C - Q \times P$
3. While $R > P$ do $R = R - P$ (at most 3 subtractions)

Output: $R \equiv C \bmod P$

Complexity : $2M(n)$ for modular reduction or $3M(n)$ for modular multiplication (assuming divisions by $\beta$ are free)

# Montgomery's reduction algorithm

Let $0 < P < \beta^n$ with $(P, \beta) = 1$ and $0 < C < P^2$

($C$ may be the result of a multiplication of $A < P$ and $B < P$)

Algorithm

1. Compute the smallest $Q$ s.t. $C + QP$ is a multiple of $\beta^n$

   $$Q = \mu \times C \bmod \beta^n, \quad \text{where } \mu = -1/P \bmod \beta^n \text{ is precomputed}$$

2. Compute $R = (C + Q \times P)/\beta^n$ (exact division)
3. If $R > P$ then $R = R - P$

Output: $R \equiv C\beta^{-n} \bmod P$

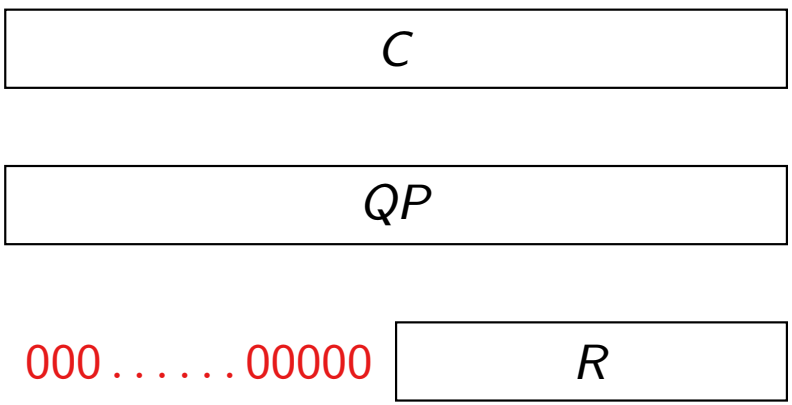
Complexity : $2M(n)$ for modular reduction or $3M(n)$ for modular multiplication (assuming divisions by $\beta$ are free)

# Barrett vs Montgomery

Barrett (MSB algorithm)

Precomputation: $\lfloor \beta^{2n}/P \rfloor$

Complexity: $2M(n)$

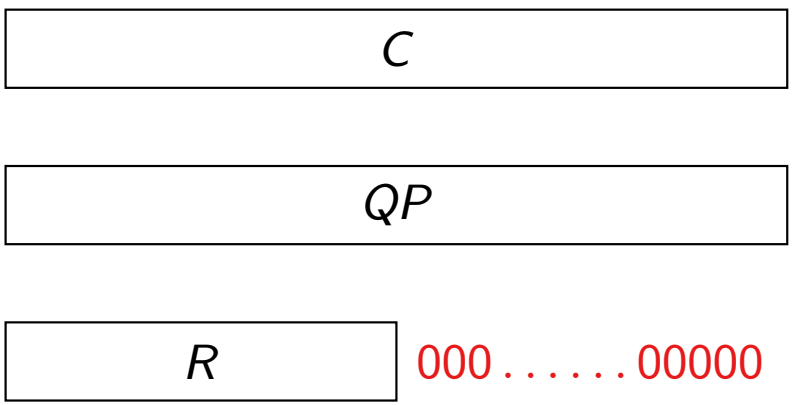



$$R = C - QP$$

Montgomery (LSB algorithm)

Precomputation: $-1/P \bmod \beta^n$

Complexity: $2M(n)$




$$R = (C + QP)\beta^{-n}$$

# Parallelization of Barrett/Montgomery's multiplication

## Barrett

$$C = A \times B$$

$$Q = \lfloor \lfloor C/\beta^n \rfloor \times \nu/\beta^n \rfloor$$

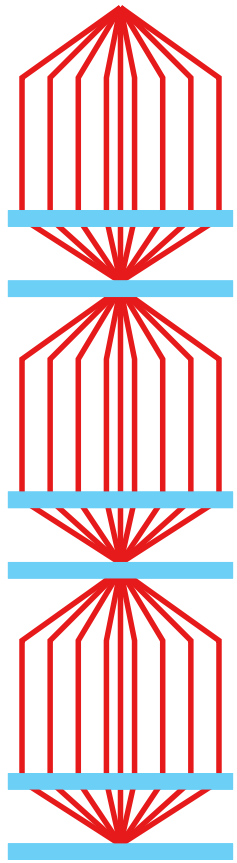
$$R = C - Q \times P$$

## Montgomery

$$C = A \times B$$

$$Q = \mu \times C \bmod \beta^n$$

$$R = (C + Q \times P)/\beta^n$$



$3M(n/\theta_1, n/\theta_2)$
on $\theta_1\theta_2$ cores

6 thread synchro

# Estimating parallelizm overhead

We used tools from "The EPCC OpenMP microbenchmarks v3.0"

$T_s$: sequential time for a portion of code

$T_p$: time for the parallel version of this on $p$ processors

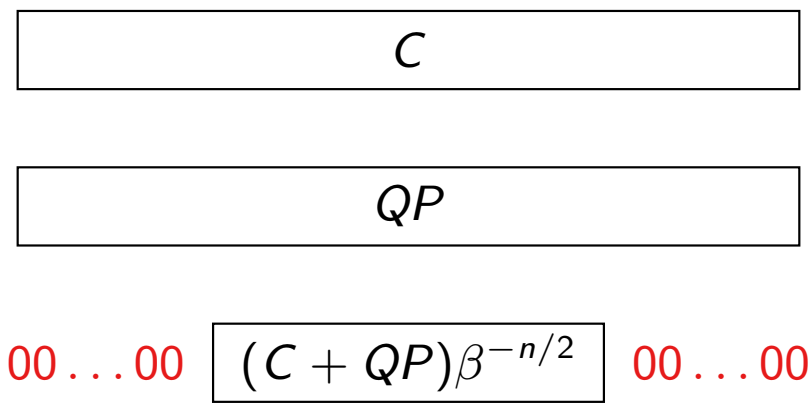
Overhead: $O_p = T_p - T_s/p$

Estimates on Intel Xeon X5650 Westmere at 2.66 GHz

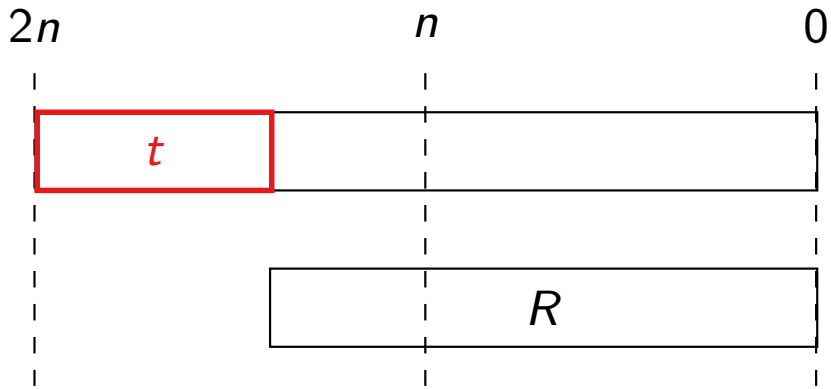
927 cycles for 2 threads
1303 cycles for 3 threads
1559 cycles for 4 threads
2221 cycles for 6 threads
2571 cycles for 8 threads

# Kaihara and Takagi bipartite's algorithm (2008)

Idea: reduce the $n/2$ MSB using a classical division or a (partial) Barrett reduction and the $n/2$ LSB using a (partial) Montgomery reduction



## **PBR**: Partial Barrett's Reduction

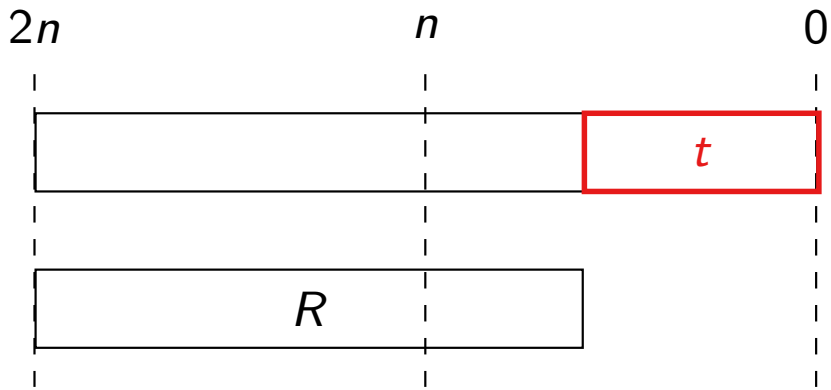



$$R \equiv C \bmod P$$

$$R < \beta^{2n-t} \quad (\text{for } t \leq n)$$

$$M(t) + M(n, t)$$

## **PMR**: Partial Montgomery's Reduction

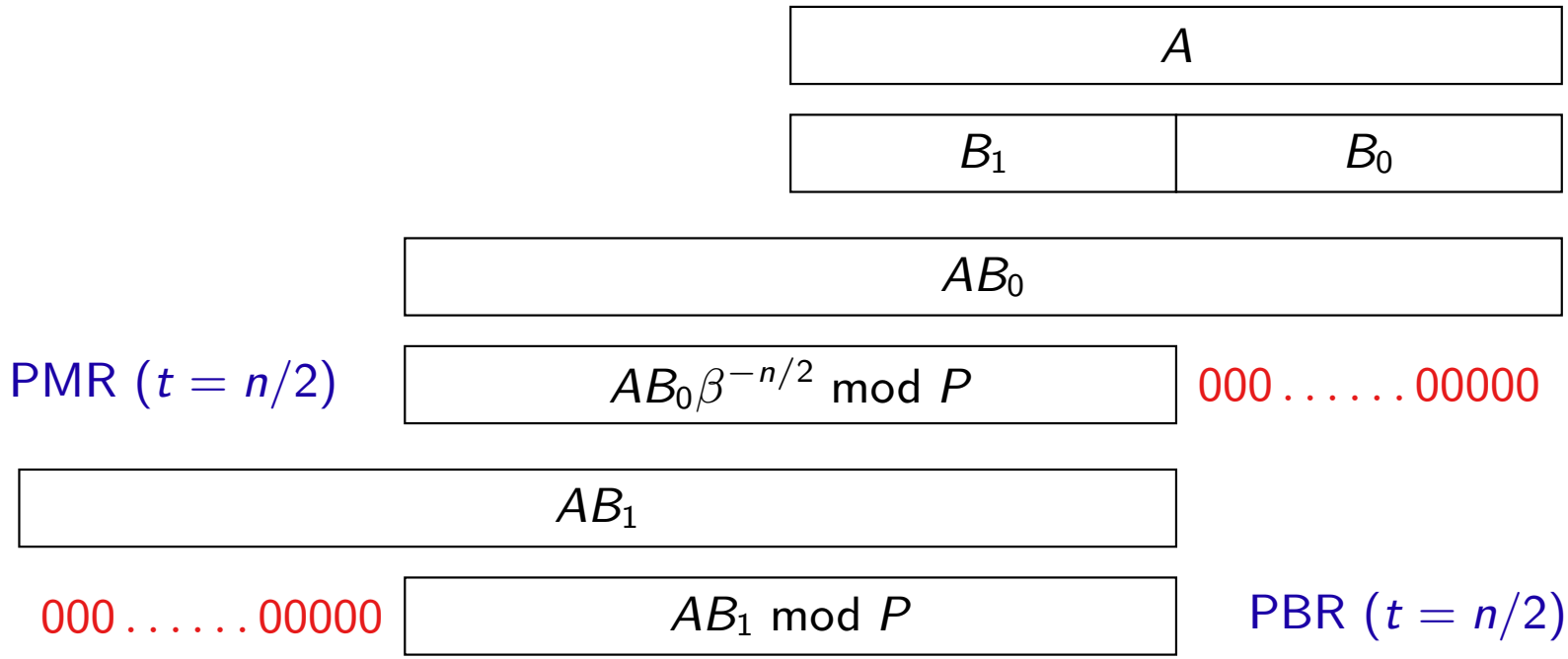



$$R \equiv C\beta^{-t} \bmod P$$

$$R < \beta^{2n-t} \quad (\text{for } t \leq n)$$

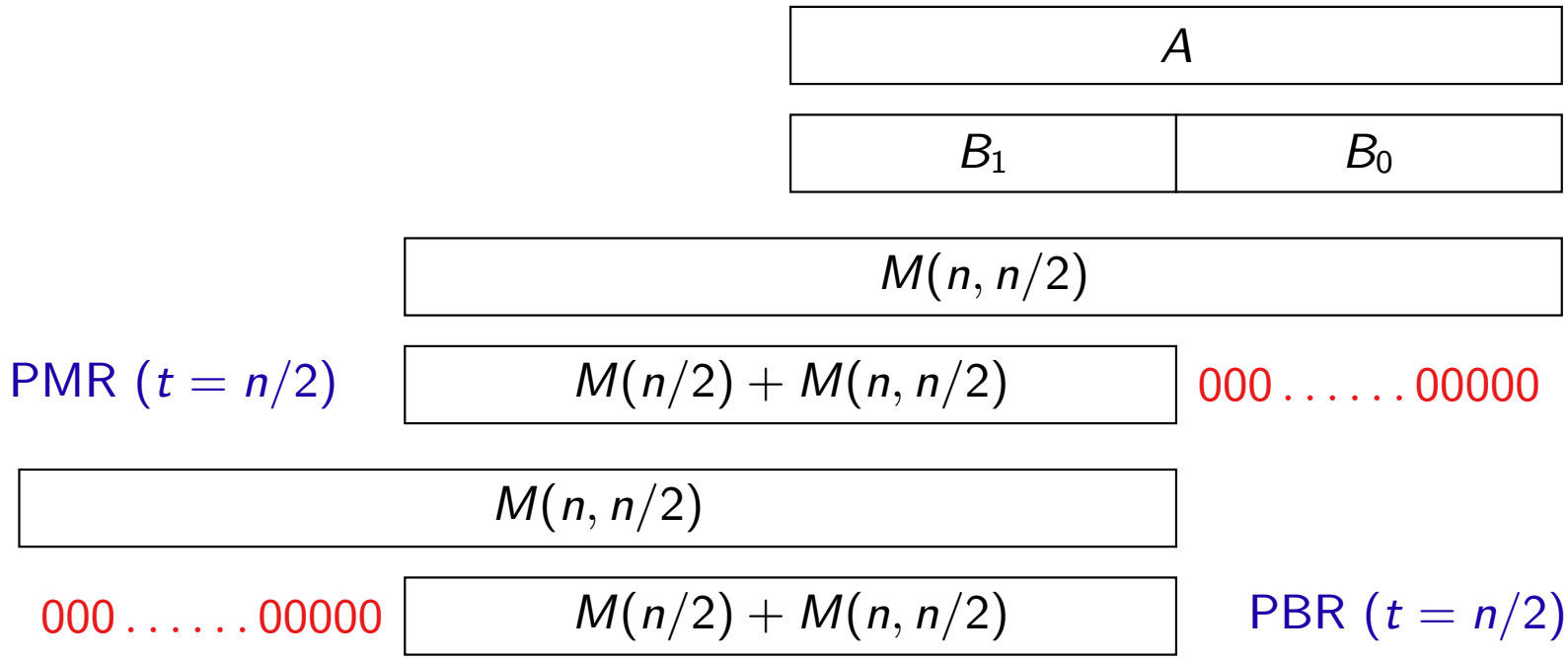
$$M(t) + M(n, t)$$

# Bipartite multiplication




$$AB\beta^{-n/2} \bmod P = (AB_1 \bmod P + AB_0\beta^{-n/2} \bmod P) \bmod P$$

# Complexity of the bipartite multiplication




- $2M(n/2) + 4M(n, n/2)$ on 1 thread
- $M(n/2) + 2M(n, n/2)$ on 2 threads, 2 sync
- $M(n/2\theta_1, n/2\theta_2) + 2M(n/\theta_1, n/2\theta_2)$ on $2\theta_1\theta_2$ threads, 6 sync

# Sakiyama *et. al.*'s tripartite multiplication (2011)

Splits both operands in two halves

Uses Karatsuba scheme and computes:

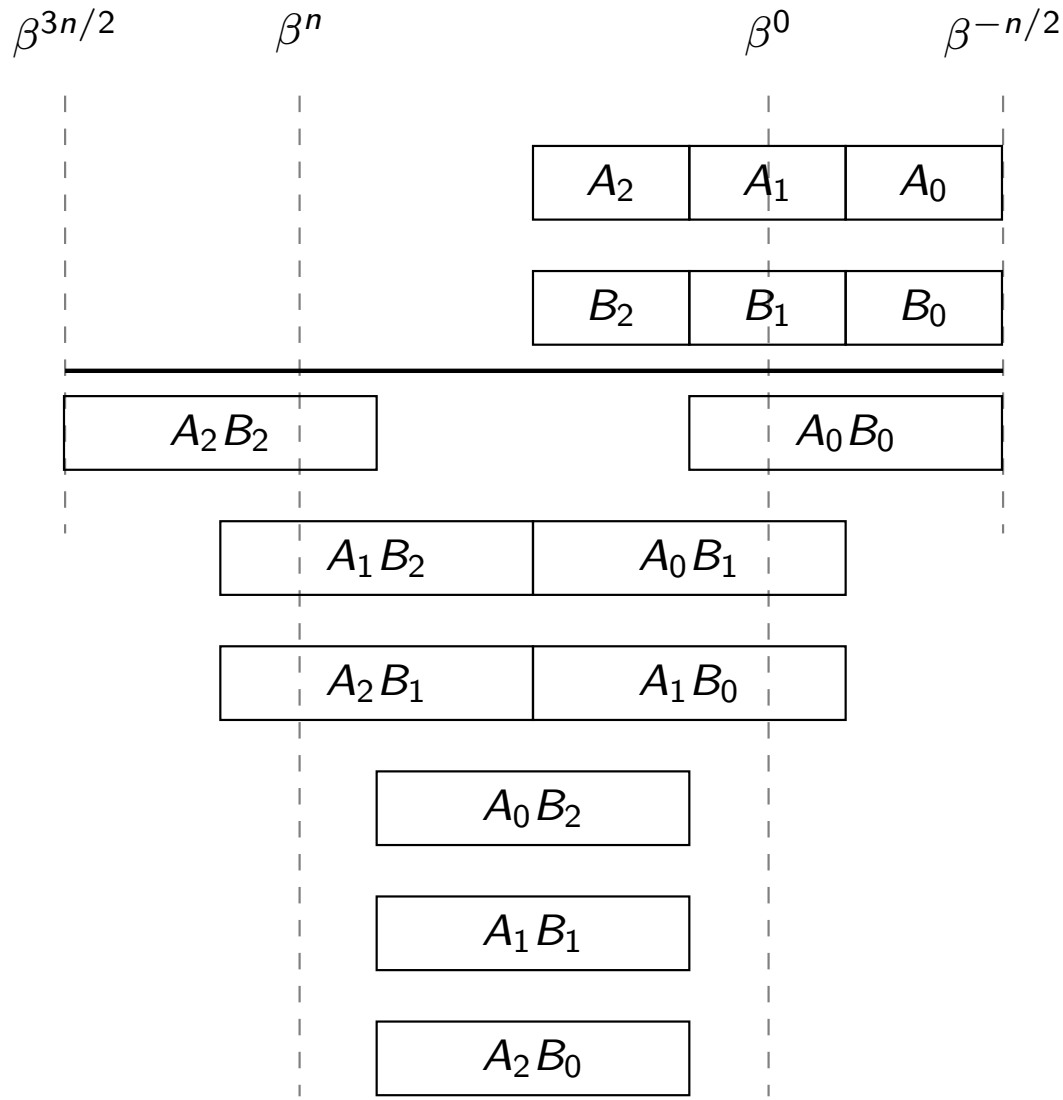
$$AB\beta^{n/2} \bmod P = \left(X_2\beta^{n/2} + (X_1 - X_2 - X_0) + X_0\beta^{-n/2}\right) \bmod P,$$

where $X_0 = A_0B_0, \quad X_1 = (A_1 + A_0)(B_1 + B_0), \quad X_2 = A_1B_1$

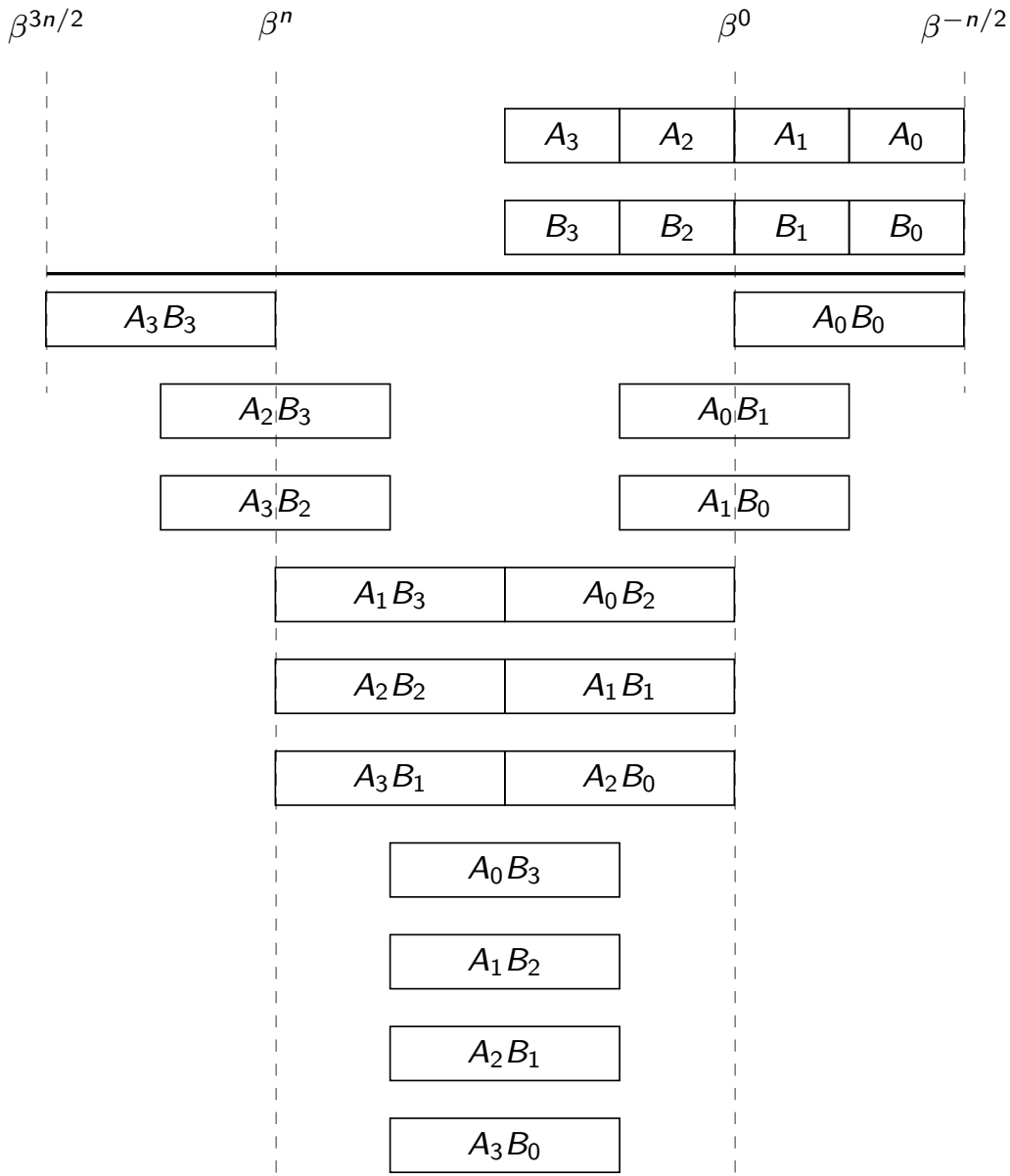
Complexity: $5M(n/2) + 2M(n, n/2)$

Too many synchronizations in our context

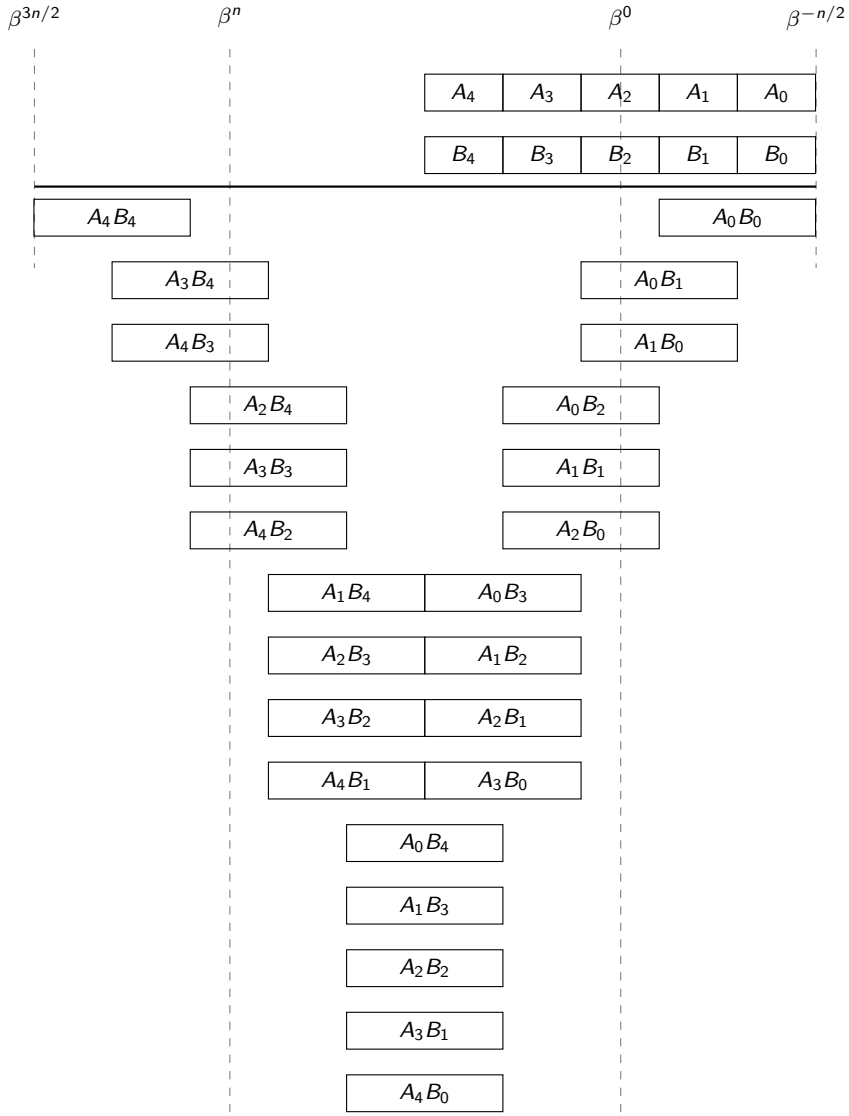
# Generalizations ($k = 3$)



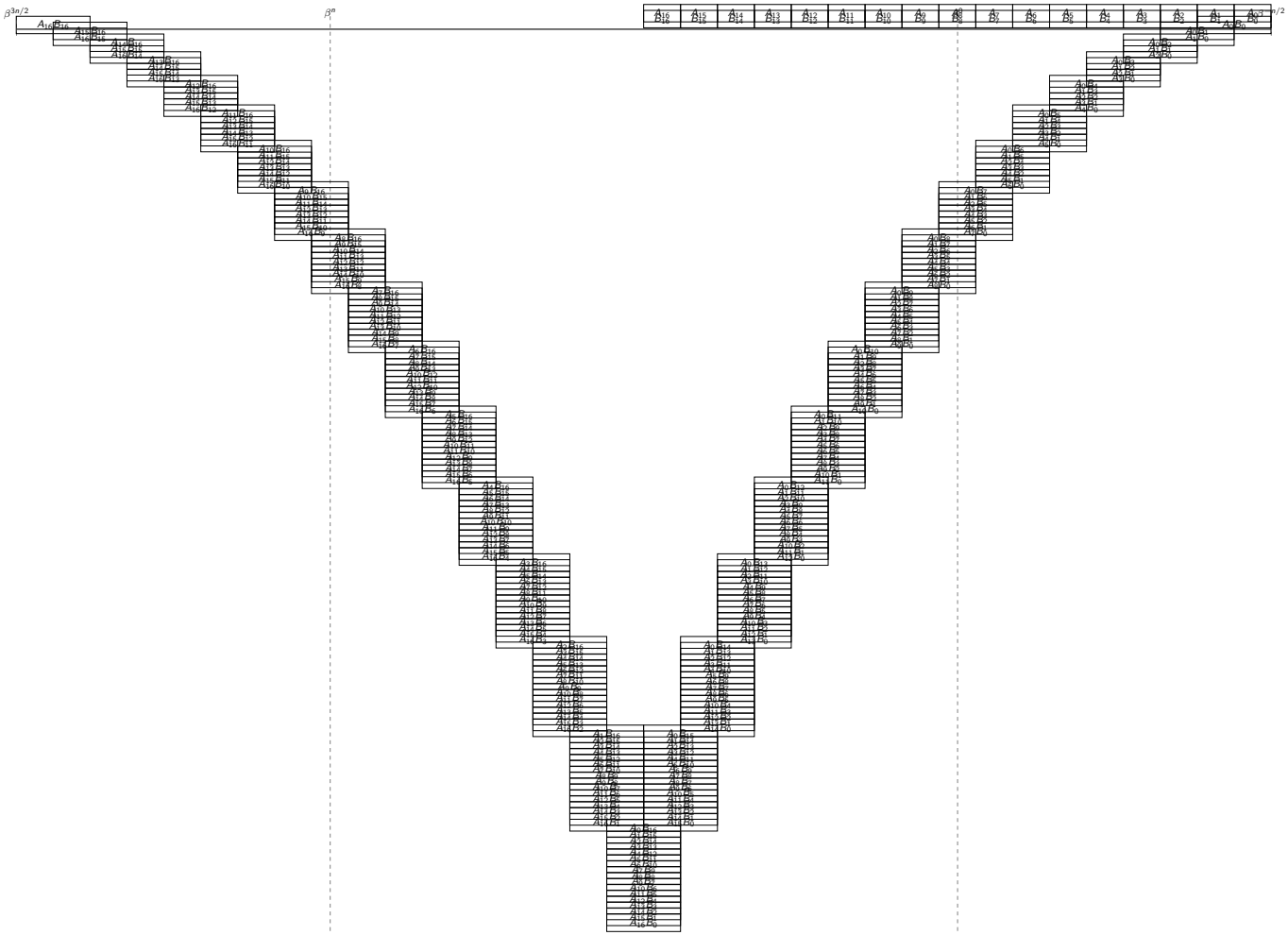
# Generalizations ($k = 4$)



# Generalizations ($k = 5$)



# Generalizations ($k = 17$)

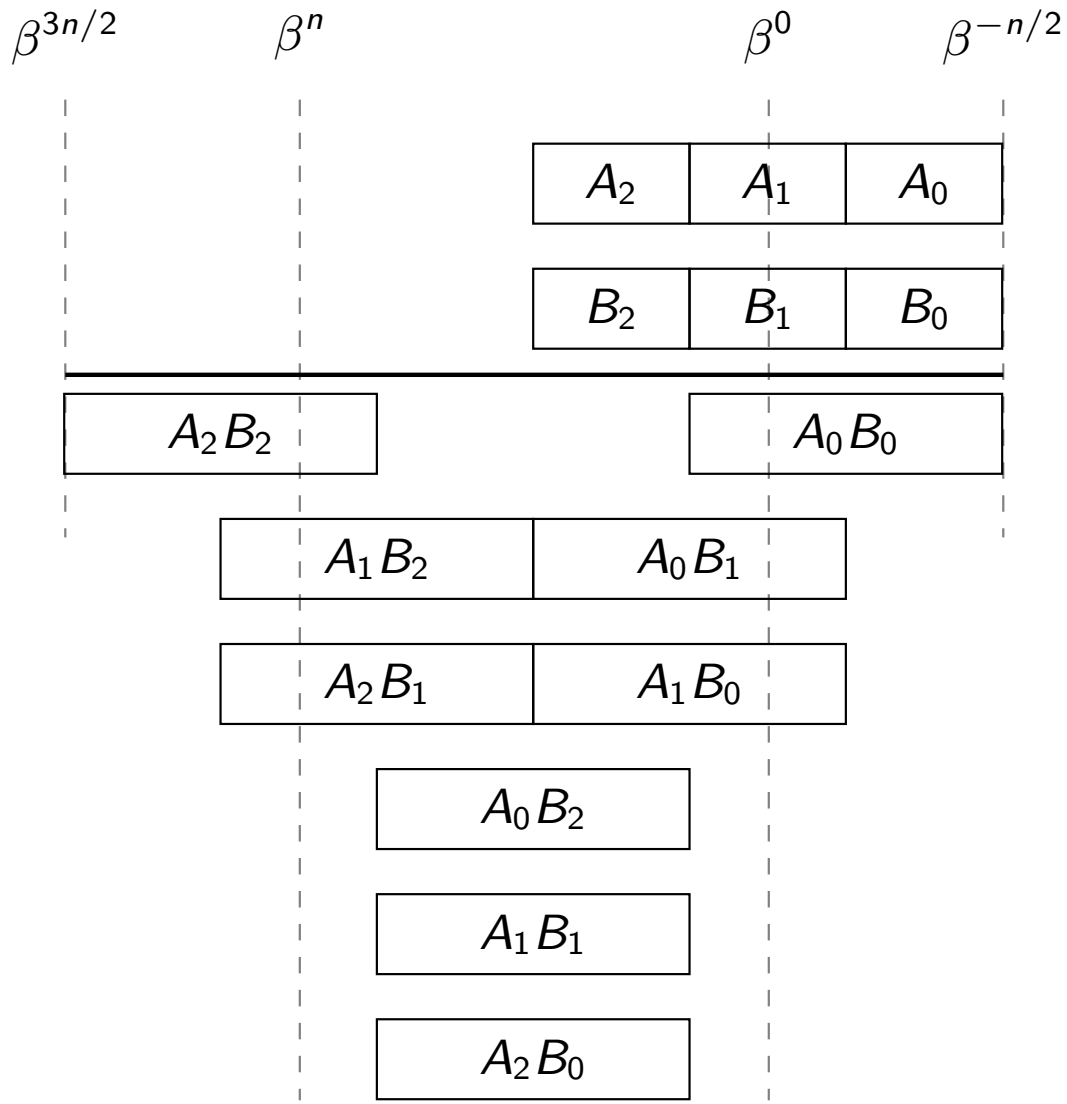


# $k$-ary multipartite modular multiplication

Keep computations independent to minimize thread synchronizations

Naive implementation requires 9 threads and is very unbalanced

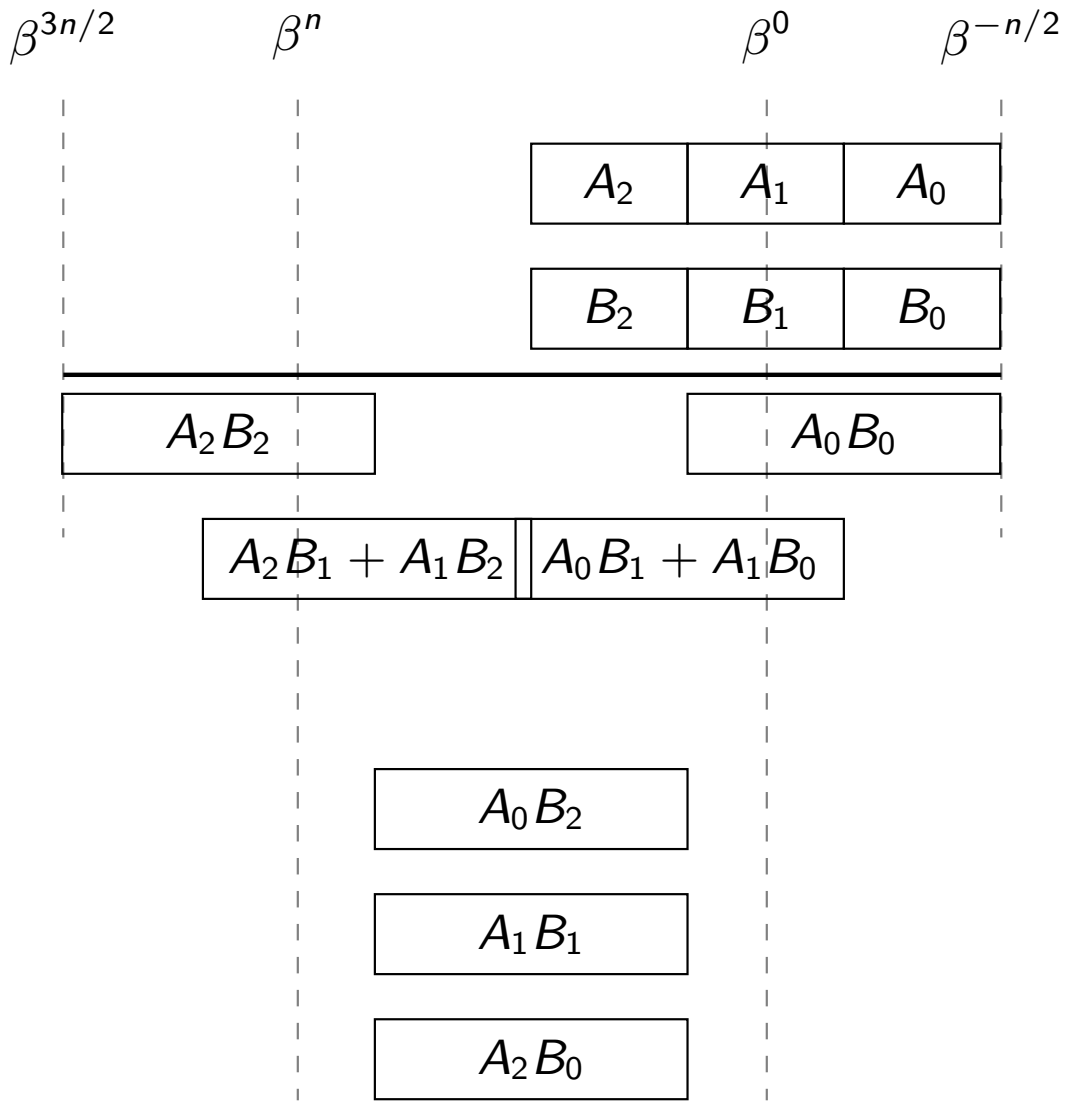
Complexity depends on #PMR and #PBR and the size of their inputs



# $k$-ary multipartite modular multiplication

Merge $Q$-values to reduce #PMR and #PBR from $k^2/4 + O(k)$ to $\approx k$

Same parallel cost with 4 threads only!



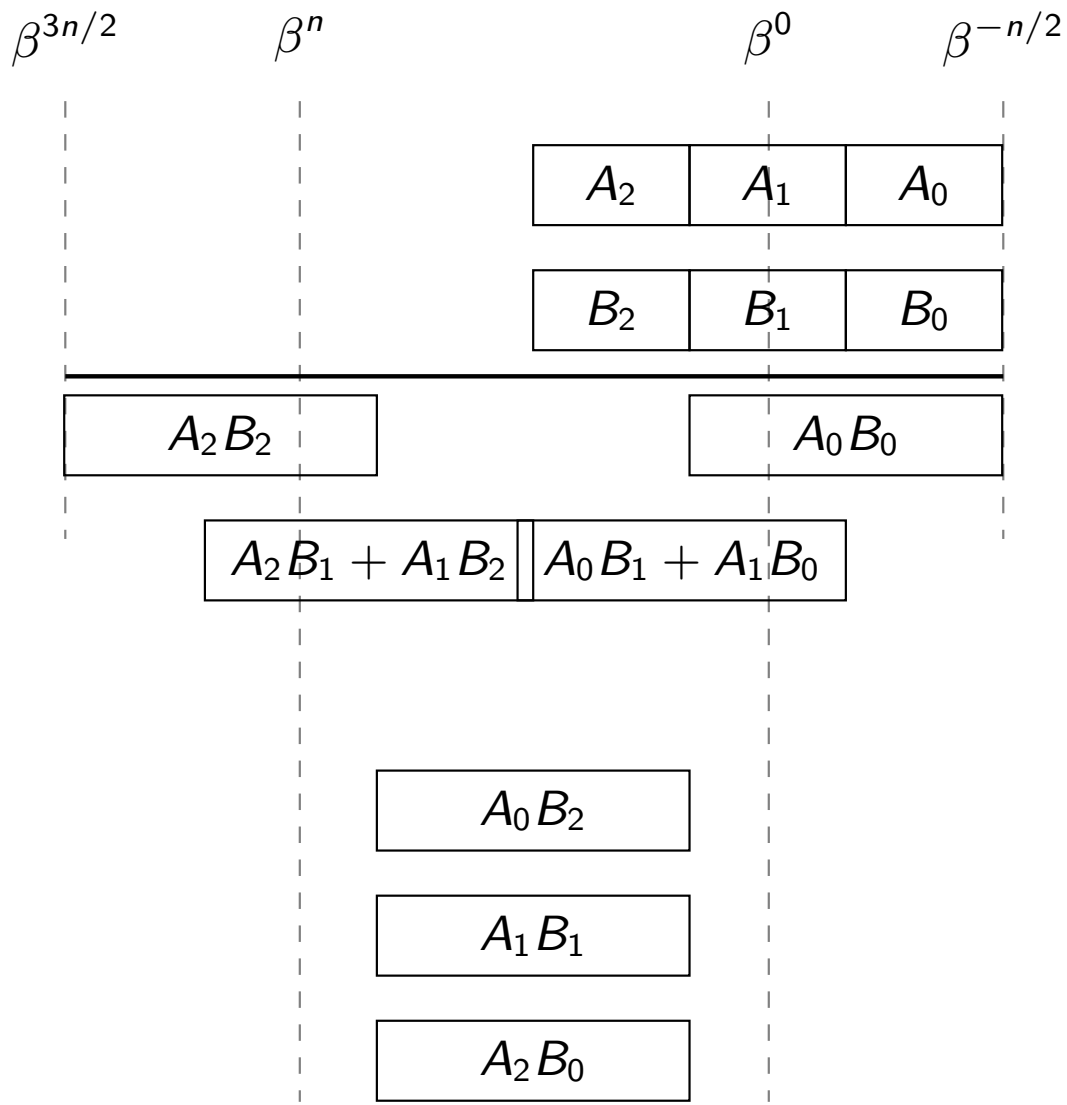
# $k$-ary multipartite modular multiplication

Assuming $M(n_1 + n_2, n) = M(n_1, n) + M(n_2, n)$

$M(n/3) + M(n/2) + M(n, n/2) = 31M(n/6)$

$2M(n/3) + M(n/6) + M(n, n/6) = 15M(n/6)$

$3M(n/3) = 12M(n/6)$



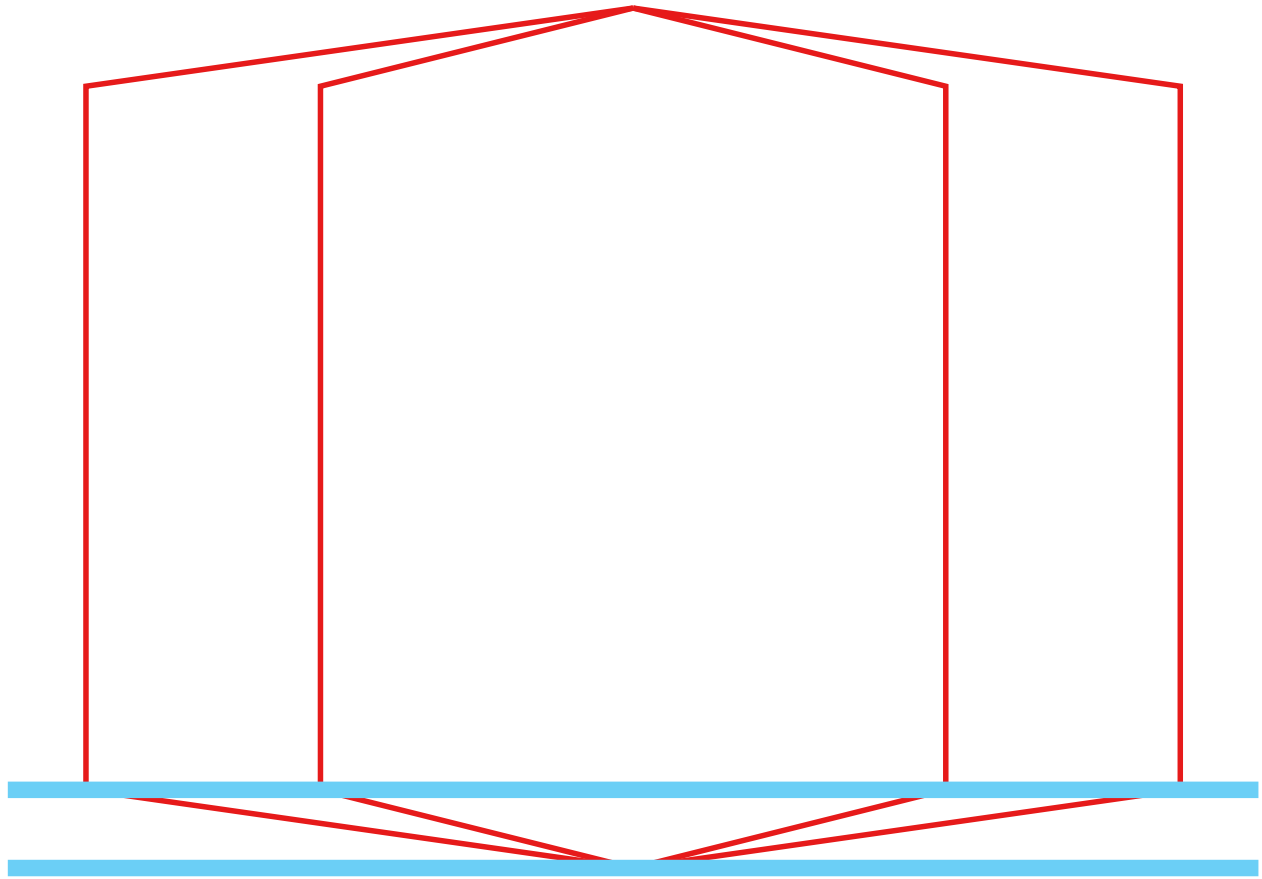
# Breaking $QP$

Postpone and merge all the multiplications by $P$ from PMR (resp. PBR) into a unique product $QP$ done in parallel at the end

$$C_d = \sum A_i B_j$$

$$R_d = \begin{cases} \text{PMR} \\ \text{PBR} \\ C_d \end{cases}$$

$$R = \sum R_d$$



# Breaking $QP$

Postpone and merge all the multiplications by $P$ from PMR (resp. PBR) into a unique product $QP$ done in parallel at the end

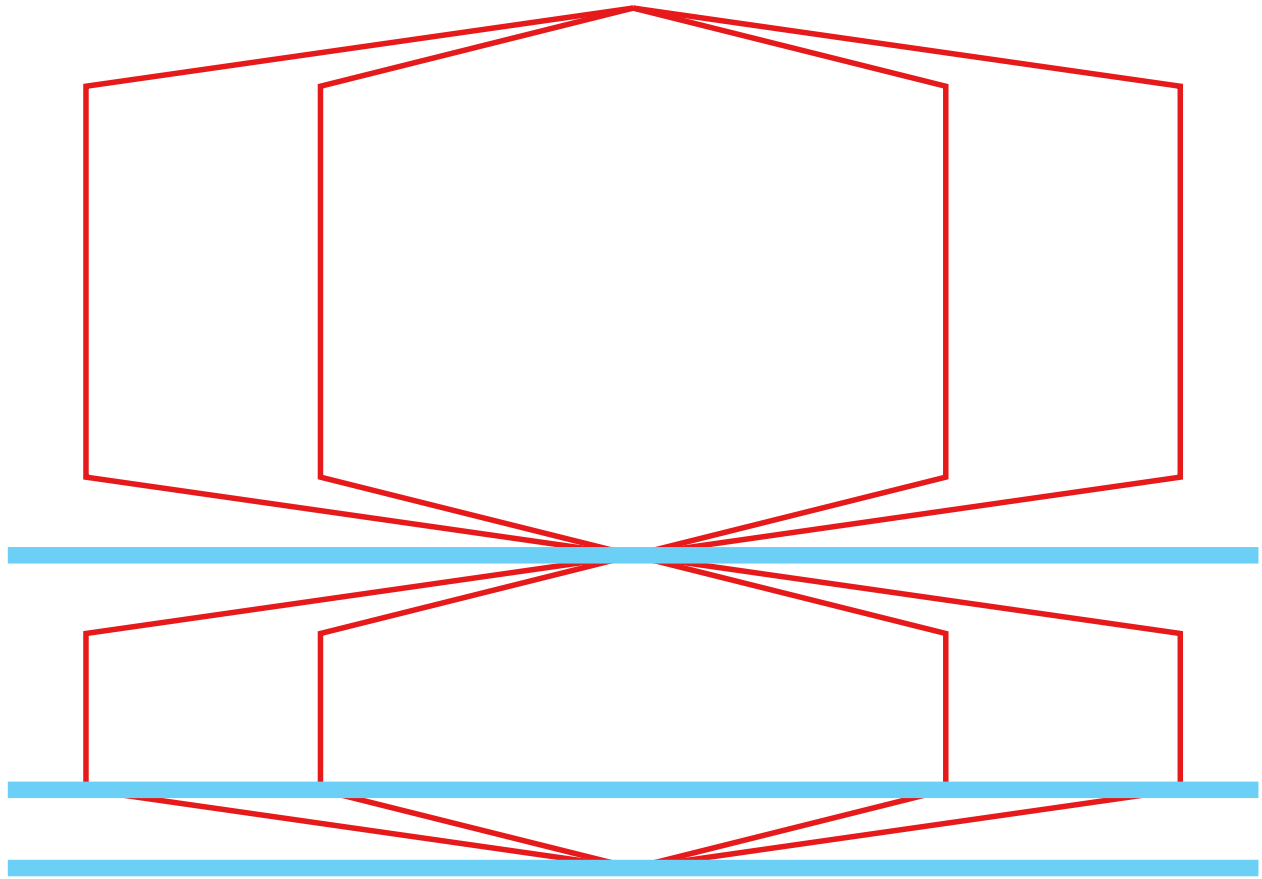
$C_d = \sum A_i B_j$

$Q_d = C_d \mu$

$Q = \sum Q_d$

$QP$ in parallel

$R = \sum C_d - QP$



# Parallel complexity

Theorem: Let us assume that $M(n_1 + n_2, n) = M(n_1, n) + M(n_2, n)$. If $T = \theta_1\theta_2$ cores are available, then using a quadratic scheme with $k > 3$ and $T \geq 3\lceil k/2 \rceil$, the parallel complexity of the $k$-ary multipartite multiplication is at most

$$M(n/k) + M(n/2, 2n/k) + M(n/\theta_1, n/\theta_2)$$

Corollary: Let $T = \theta_1\theta_2$. Let $k = 2T/3$. Then, assuming $M(n_1 + n_2, n) = M(n_1, n) + M(n_2, n)$, the parallel complexity of the $k$-ary multipartite algorithm is bounded by

$$(2.5 + 9/4\theta_1\theta_2)M(n/\theta_1, n/\theta_2)$$

and the number of thread synchronizations is exactly 3.

# Comparisons

| Algorithm | Parallel complexity on $\theta_1\theta_2$ cores | # sync. |
|---|---|---|
| Montgomery/Barret | $3M\left(\frac{n}{\theta_1}, \frac{n}{\theta_2}\right)$ | 6 |
| Bipartite | $2.5M\left(\frac{n}{\theta_1}, \frac{n}{\theta_2}\right)$ | $(*)$ 6 |
| $k$-ary multipartite v1 | $(2.5 + \frac{9}{4\theta_1\theta_2})M\left(\frac{n}{\theta_1}, \frac{n}{\theta_2}\right)$ | 3 |
| $k$-ary multipartite v2 | $(1.5 + \frac{9}{4\theta_1\theta_2} + \frac{\theta_1\theta_2}{2})M\left(\frac{n}{\theta_1}, \frac{n}{\theta_2}\right)$ | 2 |

$(*)$ when $\theta_1\theta_2 = 2$, the bipartite algorithm requires only 2 synchronizations

# Experiments

| | | sizes (in bits) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Algorithm | 1024 | 1536 | 2048 | 3072 | 4096 | 6144 | 8192 | 12288 | 16384 |
| 1 Thread | Best seq. | 1.32 | 2.53 | 4.13 | 8.18 | 13.06 | 26.03 | 41.10 | 79.76 | 125.18 |
| 3 Threads | Montgomery | 3.63 | 4.15 | 4.95 | 6.71 | 8.84 | 13.42 | 20.11 | 33.96 | 50.50 |
| | 2-ary multi. v1 | 2.59 | 3.23 | 3.83 | 5.49 | 7.43 | 12.24 | 18.26 | 32.17 | 48.40 |
| | 2-ary multi. v2 | 1.47 | 2.04 | 2.86 | 4.84 | 7.45 | 12.91 | 19.95 | 37.59 | 58.48 |
| 4 Threads | Montgomery | 3.73 | 4.34 | 4.94 | 6.54 | 9.60 | 13.05 | 19.09 | 33.09 | 49.77 |
| | Bipartite | 3.93 | 4.08 | 4.69 | 6.27 | 7.94 | 12.13 | 17.12 | 29.60 | 44.65 |
| | 4-ary multi. v1 | 2.75 | 3.05 | 3.68 | 5.06 | 7.07 | 11.40 | 17.38 | 31.04 | 48.16 |
| | 4-ary multi. v2 | 1.68 | 2.14 | 2.90 | 4.74 | 7.20 | 13.43 | 20.60 | 37.88 | 60.99 |
| 6 Threads | Montgomery | 4.66 | 5.10 | 5.71 | 6.70 | 8.72 | 12.18 | 17.30 | 26.82 | 41.94 |
| | Bipartite | 4.82 | 5.20 | 5.39 | 6.45 | 7.88 | 10.99 | 15.16 | 22.92 | 34.58 |
| | 4-ary multi. v1 | 3.32 | 3.47 | 3.83 | 5.13 | 6.56 | 9.72 | 14.48 | 24.13 | 36.22 |
| | 4-ary multi. v2 | 1.95 | 2.42 | 3.03 | 4.96 | 6.91 | 12.00 | 17.82 | 32.08 | 49.84 |
| 8 Threads | Montgomery | 7.62 | 7.99 | 8.59 | 10.51 | 13.01 | 16.39 | 20.18 | 30.59 | 42.36 |
| | Bipartite | 10.12 | 9.98 | 10.33 | 11.05 | 12.25 | 15.45 | 18.90 | 26.61 | 36.58 |
| | 8-ary multi. v1 | 5.85 | 6.03 | 6.44 | 7.57 | 8.87 | 12.18 | 16.06 | 25.43 | 37.80 |
| | 8-ary multi. v2 | 3.98 | 4.29 | 4.92 | 6.59 | 8.84 | 13.81 | 19.88 | 33.92 | 51.10 |

Timings in $\mu$s. The gray cells represent the fastest algorithms for each given size.

# Conclusions and leftovers

- Parallelization of modular multiplication may be relevant in software
- Increasing the number of cores improves the performance but full usage of the cores may not be the best strategy
- For operands larger than $2^{13}$ bits, synchronization-friendly algorithms offer better performance than those with the best complexity
- For larger inputs, bipartite is the best alternative

- Modular reduction seems more flexible
- Modular squaring (heavily used in modular exponentiation)
- Much bigger numbers, many-cores, GPUs, ...
- There seem to be some interest for parallel multiple precision FP algorithms and implementations