Systolic Methods for GCD Computation

Paul Vrbik
University of Western Ontario

May 21, 2010

The gcd problem in $\mathbb{Z}[x]$

Let:

$$A = a_0 + \cdots + a_i x^i$$
$$B = b_0 + \cdots + b_j x^j$$

where $A, B \in \mathbb{Z}[x]$.

Definition (gcd)

The $\gcd(A, B)$ is the largest common divisor of A and B.

Example

If $A = (x+1)(x+2)$ and $B = (x+2)(x+3)$ then

$$\gcd(A, B) \sim (x+2),$$

(i.e. some $\mathbb{Z}$-multiple of $(x+2)$).

A transformation

$$T : \mathbb{Z}[x] \to \mathbb{Z}[x]$$

which maps $A \in \mathbb{Z}[x]$ to $\bar{A} \in \mathbb{Z}[x]$ is a gcd preserving transformation if

$$\gcd(A, B) = \gcd(\bar{A}, B).$$

If we pick a transformation such that $\deg \bar{A} < \deg A$ then eventually we get

$$\gcd(A, B) = \gcd(0, \bar{B}) = \bar{B}.$$

The Euclidean step is a transformation given by

$$A \xrightarrow{T} A - qx^d B$$

where $d = \deg(A) - \deg(B) = i - j \geq 0$ and $q = \frac{a_i}{b_j}$ (this guarantees that the leading term of A vanishes).

Example

Let $A = x^2 + 3x + 2$ and $B = x^2 + 5x + 6$.

$$\bar{A} = (x^2 + 3x + 2) - \left(\frac{1}{1}x^{2-2}\right)(x^2 + 5x + 6) = -2x - 4$$

$$\bar{B} = (x^2 + 5x + 6) - \left(\frac{1}{2}x^{2-1}\right)(2x + 4) = 3x + 6$$

$$\gcd(A, B) = \gcd\left(-2(x+2), 3(x+2)\right) = x + 2$$

The gcd computation can be seen as a repeated application of the transformation T:

$$A \xrightarrow{T} \bar{A}_0 \xrightarrow{T} \cdots \xrightarrow{T} \bar{A}_n \xrightarrow{T} 0$$
$$B \xrightarrow{T} \bar{B}_0 \xrightarrow{T} \cdots \xrightarrow{T} \bar{B}_n \xrightarrow{T} \gcd(A, B)$$

Systolic Arrays

Definition (systolic array)

A pipe network arrangement of processing units called CELLS (i.e. processors) with no communication (each cell has it's own memory).

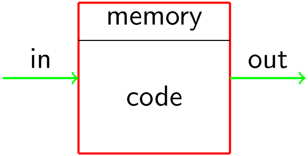


Figure: A systolic cell.

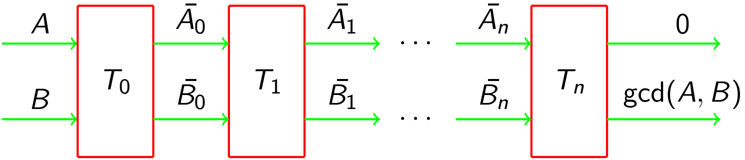


Figure: A systolic array for gcd computation.

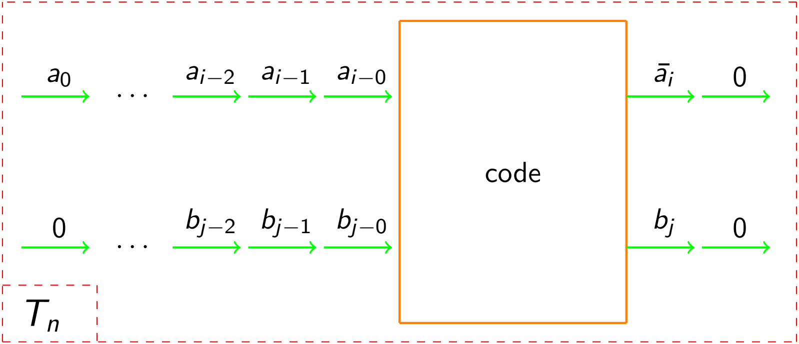
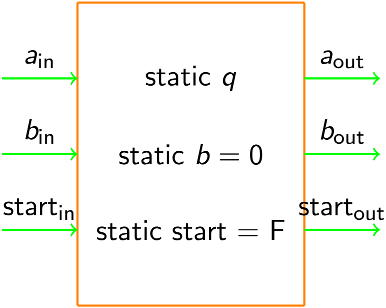


Figure: A systolic cell which takes terms instead of polynomials.

The key feature of this systolic cell is that the leading (non-zero) terms of $\bar{A}$ and B come out at the same time (more on this soon).

Single Cell Design




```
if start then
  q := ain / bin;
  aout := 0
else
  aout := ain - q*bin;
end if

bout := b;
b := bin;
startout := start;
start := startin;
```

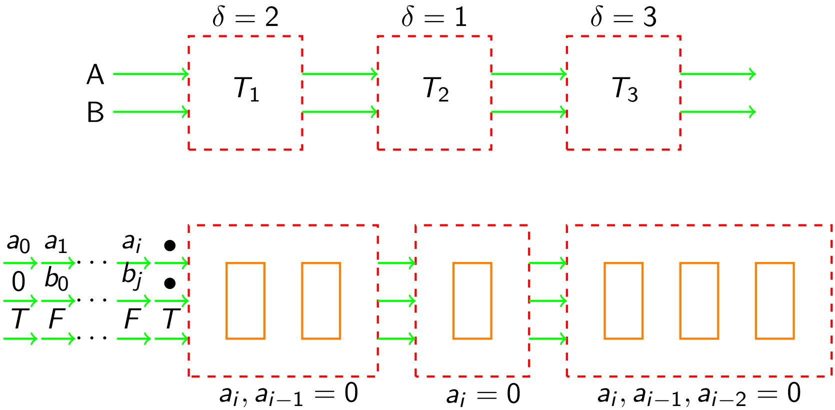
Let $\delta = \deg(A) - \deg(\bar{A})$ (δ is a measure of how much A was reduced). Typically $\delta = 1$ but sometimes it's more than one. We call δ_α the reduction value of T_α.

Since the total degree of A_0 and B_0 is $i + j$, the sum of reduction values obeys

$$\sum_{\alpha=1}^{k} \delta_\alpha \leq i + j + 1,$$

where k is the number of T_α's required to calculate the gcd.

Amazingly, the systolic cell given on the page before works for all cases! In fact, δ doesn't need to be calculated or even known.


δ = 2
δ = 1
δ = 3
A
B
T1
T2
T3
a0 a1 ... ai
0 b0 ... bj
T F ... F T
ai, ai−1 = 0
ai = 0
ai, ai−1, ai−2 = 0

Implementing Systolic Arrays

As hardware — One can build single purpose hardware that directly realizes the systolic array. Provided one builds enough cells this would work very quickly (linear).

On GPU — As a gpu is *many* cores embedded on a chip. One could divide the pipe's cells onto these cores and *simulate* the pipe by having each chip simulate multiple cells.

On multi-core — Seems that the memory overhead will dominate the procedure.

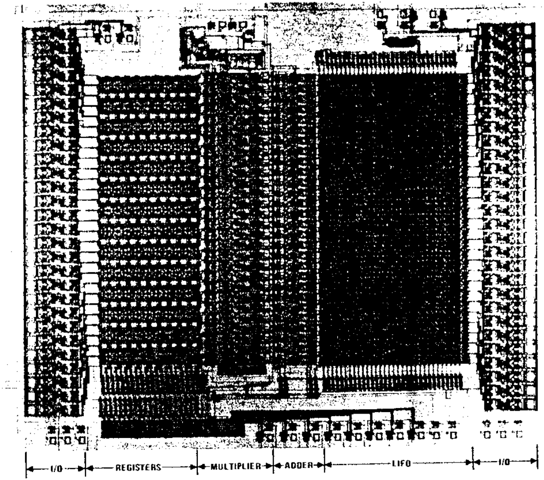
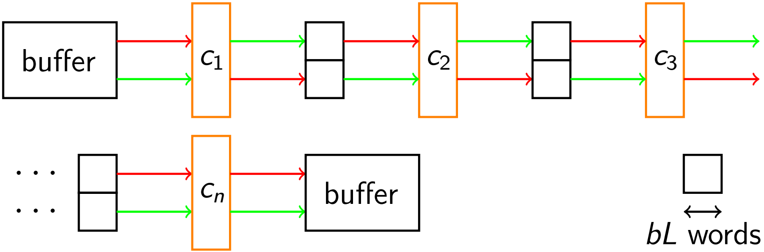


Figure: A chip layout of Toeplitz system solver.

Systolic simulation on GPU's and multicore machines.



Each core c_i simulates c cells, c is small, say $c = bL$.

Step 1

c_1 reads from $\rightarrow$, executes program $\mathbb{P}$ c times, writes to $\rightarrow$.

Step 2

c_1 reads from $\rightarrow$, executes program $\mathbb{P}$ c times, writes to $\rightarrow$.

c_2 reads from $\rightarrow$, executes program $\mathbb{P}$ c times, writes to $\rightarrow$.

Step 1

c_1 reads from $\rightarrow$, executes program $\mathbb{P}$ c times, writes to $\rightarrow$.

Step 2

c_1 reads from $\rightarrow$, executes program $\mathbb{P}$ c times, writes to $\rightarrow$.

c_2 reads from $\rightarrow$, executes program $\mathbb{P}$ c times, writes to $\rightarrow$.

Step 3

c_1 reads from $\rightarrow$, executes program $\mathbb{P}$ c times, writes to $\rightarrow$.

c_2 reads from $\rightarrow$, executes program $\mathbb{P}$ c times, writes to $\rightarrow$.

c_3 reads from $\rightarrow$, executes program $\mathbb{P}$ c times, writes to $\rightarrow$.

$\vdots$

Step p

$c_1, \ldots, c_{p-1}$ read/write from (say) $\rightarrow$.

c_p writes to big buffer.

When c_1 empties its input buffer switch it with c_p's output buffer.

Complexity Analysis

Let Y be the *minimum* number of cells required to calculate the gcd.

Let W be the *maximum* input size (in words) of the systolic system.

Definition (work)

Our measure of work will be the number of words read during the execution of the algorithm (that is, a systolic cell does zero/negligible work).

Therefore the total work is at exactly $Y \cdot W$ (each cell reads all the input).

We count the "work" done at each step i where $1 \leq i \leq s$.

$1 \leq i \leq p-1$ (not all processors running yet) we do $i \cdot c$ work.

$p+1 \leq i \leq s-p$ (all processors running) we do $p \cdot c$ work.

$s-p+1 \leq i \leq s$ (processors emptying) we do $(s-i+1) \cdot c$ work.

In total this is

$$c \sum_{i=1}^{p} i + cp \sum_{i=p+1}^{s-p} + c \sum_{i=s-p+1}^{s} (s-i+1) = cp(p+s+1)$$

Therefore, in our simulation we must have

$$YW \leq cp(p+s+1).$$

Or. in other words, we must read as many words as the theoretical model.

Example

Suppose we want $s = 2p - 1$ and $\frac{W}{c} = p$. Then we must have

$$YW \leq 3p^2c$$

and

$$W = pc$$

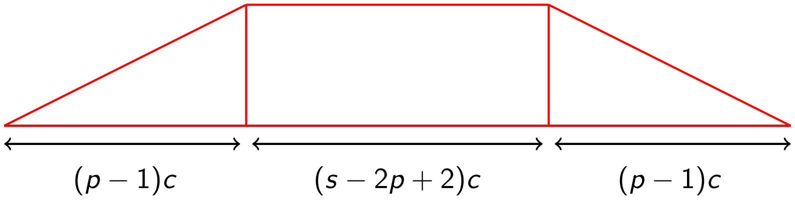
which implies that

$$Y \leq 3p$$

which is far too restrictive for Y.

Moreover, this would yield a parallelism limited to $p/2$ as half the processors will be idle half of the time.

Parallelism



We assume $s \neq -1 \bmod p$ which guarantees that the first $p-1$ and last $p-1$ steps are symmetric (and uses half of the processors on average).

Then, the parallelsim of our *Systolic Array Simulator* is

$$\frac{2 \cdot p/2(p-1)c + p(s-2p+2)c}{cs} = \frac{p}{s}(s-p+1)$$

Meaning if we want the parallelism to be at least αp for some $0 < \alpha \leq 1$; then we must have

$$s \geq \frac{p-1}{1-\alpha}.$$

Example

For $p = 4$ and $\alpha = 90\%$ we must have $s \geq 30$.

Recall that s is also constrained by

$$YW \leq cp(p + 1 + s)$$

where c is expected to be something like bL where b is the number of words read by the program $\mathbb{P}$ in one of its cycles.

The challenges of the *Systolic Array Simulator* are

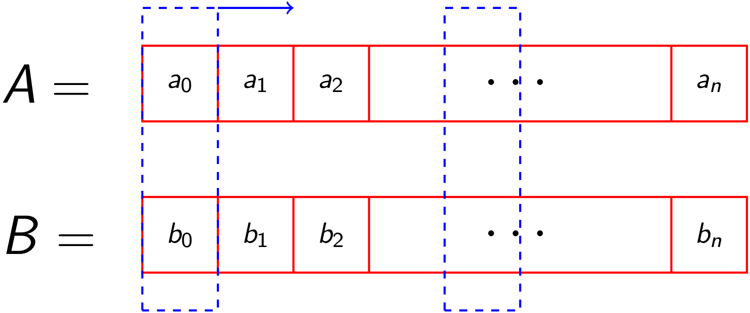
- The **if** statements in program $\mathbb{P}$
- The synchronization of the cores
- All cores need to share a fast memory (e.g. L2 cache)

We anticipate good performances on architectures like the i7 and GPUs

Concluding remarks:

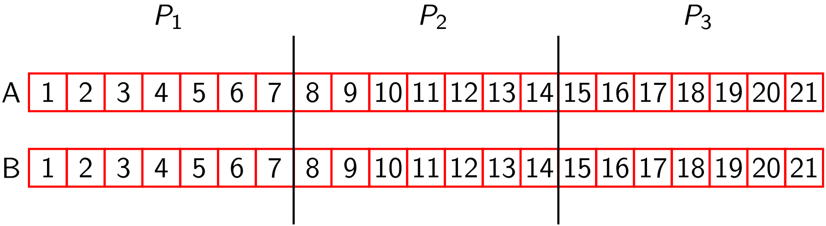
1. To increase parallelism *increase* the ratio W/c and s.
2. Will work best when p is large.
3. GPU's is the hardware best suited for this.

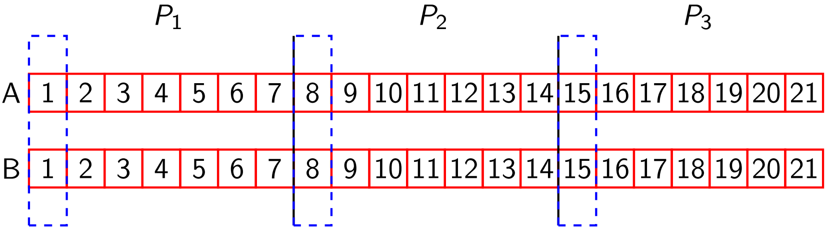
A GCD ALGORITHM FOR MULTICORE SYSTEMS?

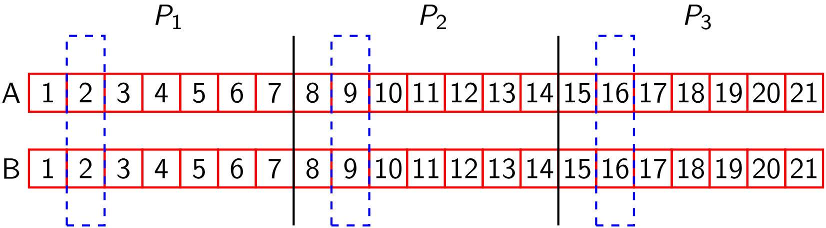


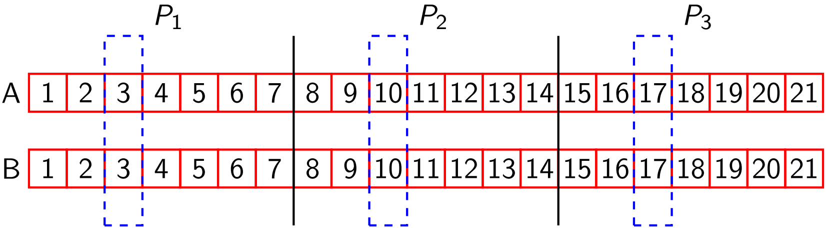

```
for i from 0 to n do
    A[i] = A[i] -qB[i];
end do;
```

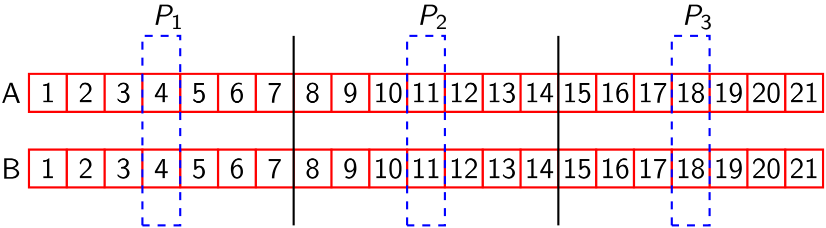
Why not do this scan in parallel?

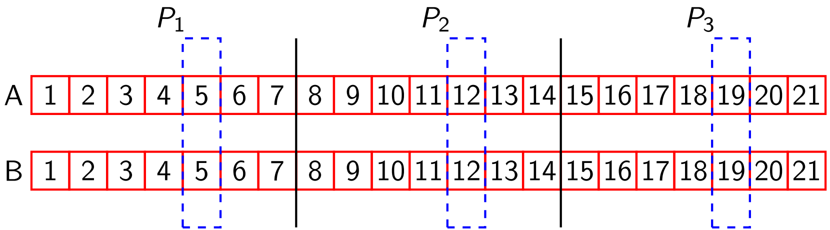

P_1
P_2
P_3
A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
B 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

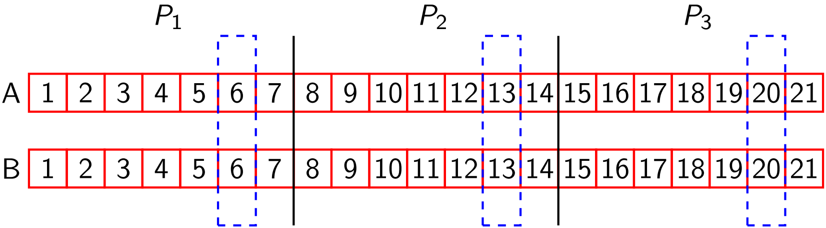

P_1
P_2
P_3
A
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
B
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

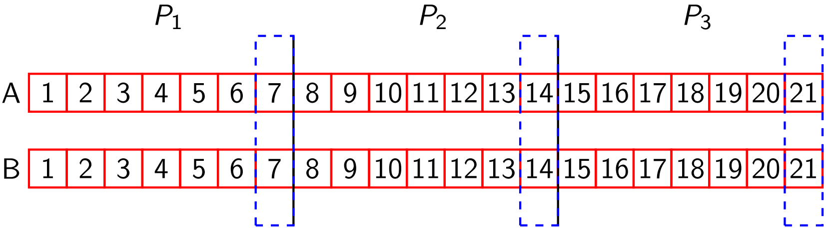

P_1
P_2
P_3
A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
B 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21


P1
P2
P3
A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
B 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21


P_1
P_2
P_3
A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
B 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

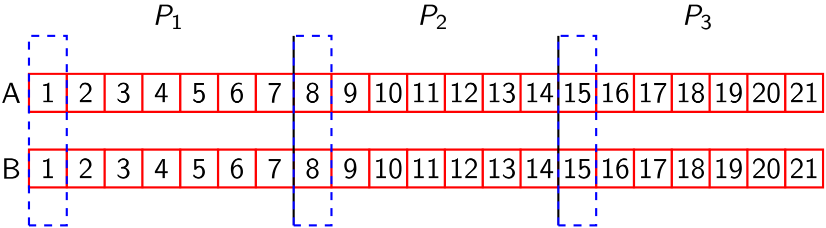

P_1
P_2
P_3
A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
B 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

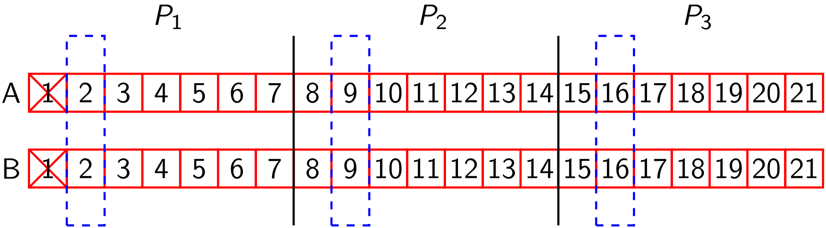

P1
P2
P3
A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
B 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

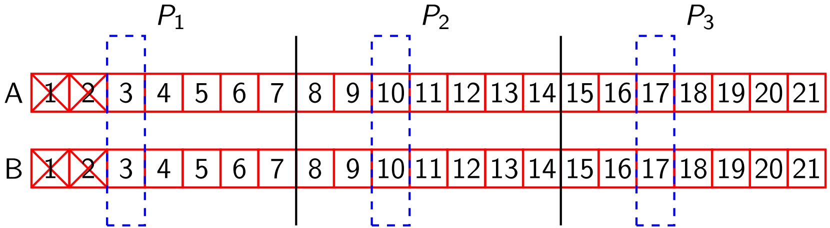

P_1
P_2
P_3
A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
B 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

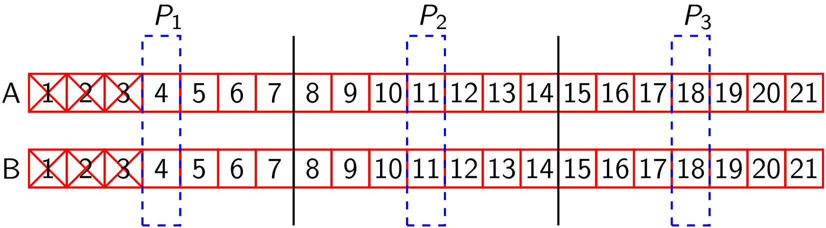
This is actually not a very good idea. Remember that the leading term of A (and B) will (by design) vanish.

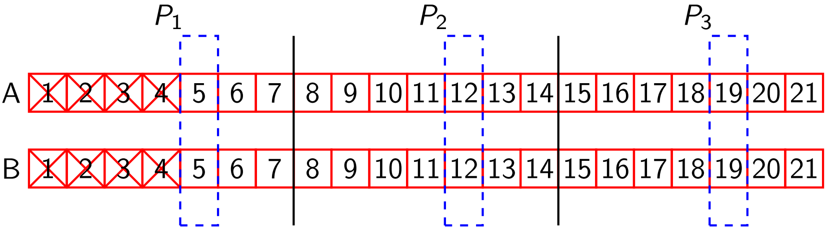
Lets take another look at the animation and this time note the vanishing terms.

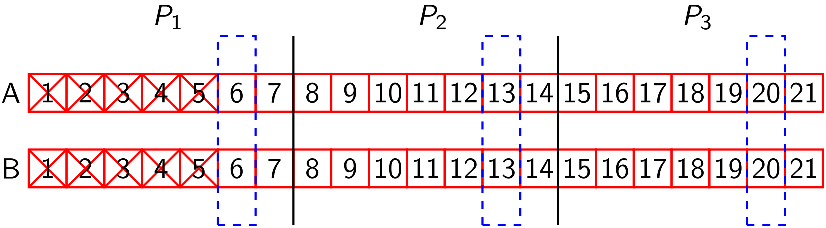

P_1
P_2
P_3
A 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
B 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

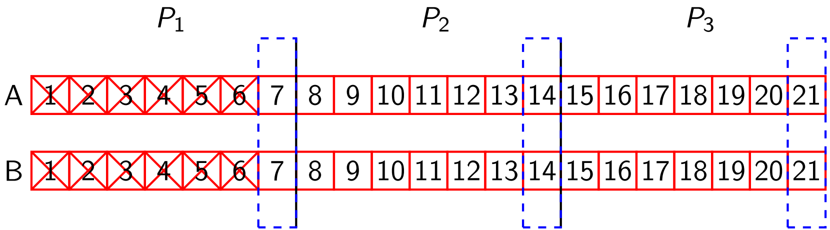

P_1
P_2
P_3
A
B
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

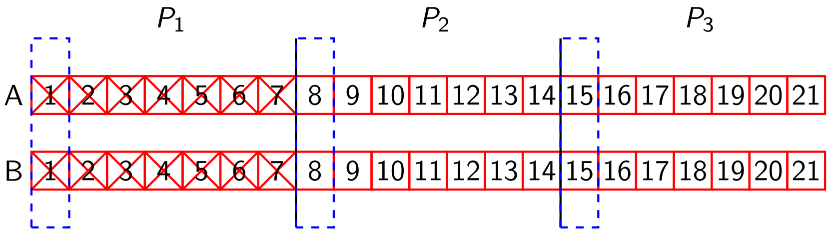

P_1
P_2
P_3
A
B
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21


P1
P2
P3
A
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
B
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21


P_1
P_2
P_3
A
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
B
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

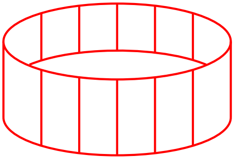
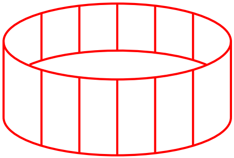

P_1
P_2
P_3
A
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
B
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21


P_1
P_2
P_3
A
B
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

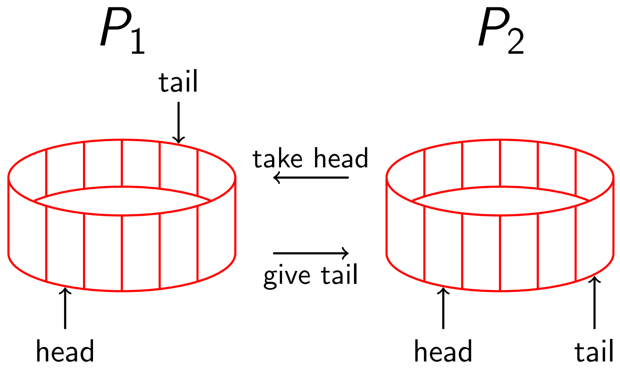

P1
P2
P3
A
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
B
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

Ribbons

P_1 P_2

$A =$  

Ribbons

$A =$ 

Now we are guaranteed that each processor will do an equal amount of work.

However, there are still some other problems:

1. How can we find the leading term of A and B?
2. How to calculate the degree of A and B?
3. How is q shared?

Q: How can we find the leading term of A and B?

A: Is just the coefficient at `head` on P1.

Q: How do we calculate the degree of A and B?

A: The number of non-empty positions (or the MAGNITUDE) of a ribbon is

$$|\text{head} - \text{tail}|.$$

The degree of a polynomial is the sum of the magnitudes of all of its ribbons.

Q: How is q shared?

A: After broadcast; synch.

- Parallelizing the gcd problem is not trivial.
- Systolic arrays (an old idea) may have found new life with GPU hardware.
- Maybe ribbons will do the job (future work).