POST-SIEVING ON GPUs

Andrea Miele[1],
Joppe W. Bos[2],
Thorsten Kleinjung[1],
Arjen K. Lenstra[1]

[1]LACAL, EPFL, Lausanne, Switzerland
[2]NXP Semiconductors, Leuven, Belgium

NUMBER FIELD SIEVE (NFS)

- Asymptotically fastest known factoring algorithm
- RSA 768-bit modulus factored with NFS in 2010
- Idea: to factor an odd composite n, find solutions

$$x, y : x^2 \equiv y^2 \bmod n \text{ and } x \not\equiv \pm y \bmod n$$

- Two main steps:
  <u>RELATION COLLECTION</u>: find smooth integers ≈90%T
  LINEAR ALGEBRA STEP: find solutions (x,y) ≈10%T

NFS RELATIONS

- Two positive integer smoothness bounds: B_r , B_a
- Irreducible $f_r(X)$, $f_a(X)$ of degree 1 and d small (d=5,6)

- Relation: (a,b) with a,b coprime integers (b>0) such that

1. $bf_r(a/b)$ is B_r-smooth except $\leq$ 3 primes > B_r and $\leq B_L$
2. $b^d f_a(a/b)$ is B_a-smooth except $\leq$ 4 primes > B_a and $\leq B_L$

COLLECT RELATIONS

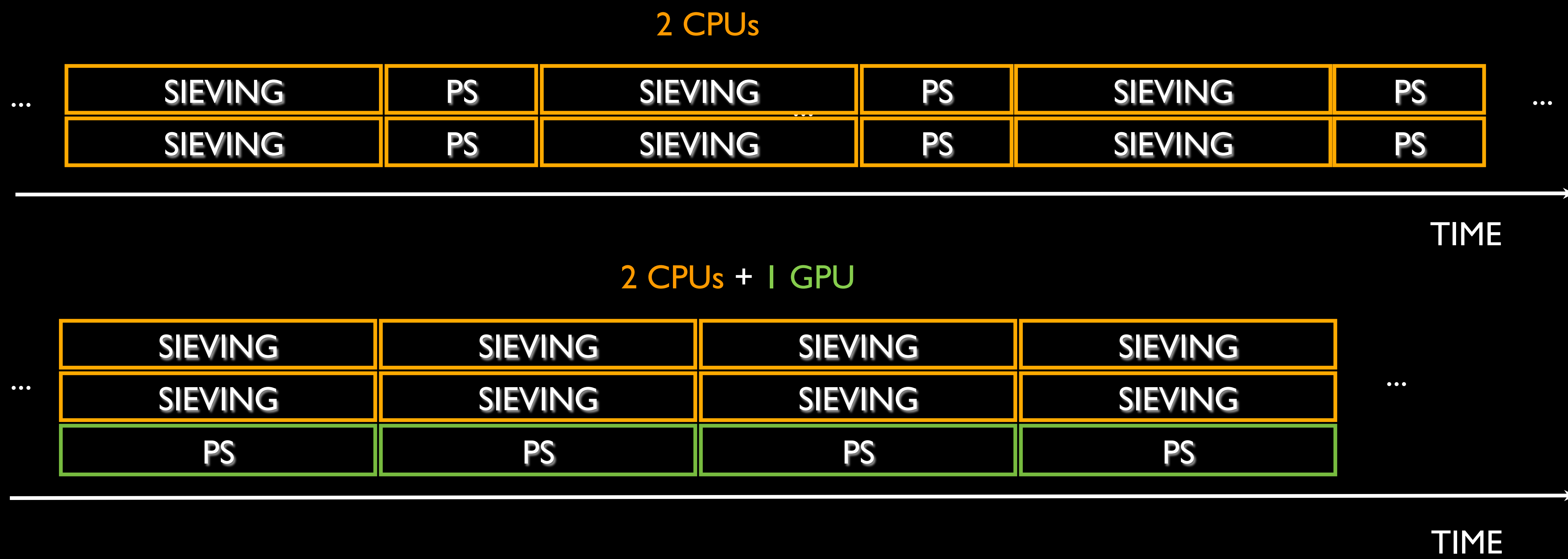
SIEVING: find pairs (a,b) s.t. $bf_r(a/b)$ ($b^d f_a(a/b)$) is product of B_r - smooth (B_a - smooth) part and "small" cofactor $\leq B_L^3$ (B_L^4)

POST SIEVING (NORMALLY 12-17% OF THE TOTAL TIME):
1 Compute $bf_r(a/b)$ and $b^d f_a(a/b)$
2 Remove small factors pair-by-pair (or re-sieve)
3 Factor cofactors pair-by-pair (COFACTORING)

EMBARRASSINGLY PARALLEL!

FASTER NFS WITH GPUs?

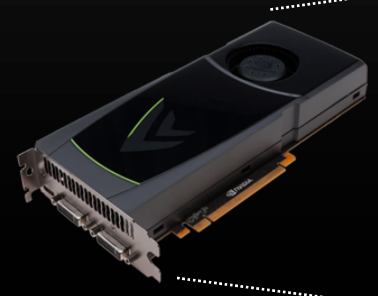
- SIEVING: memory hungry, done on CPUs
- PREVIOUSLY: offload ECM to GPUs or FPGAs
- IDEA: offload ALL POST SIEVING TO GPUs

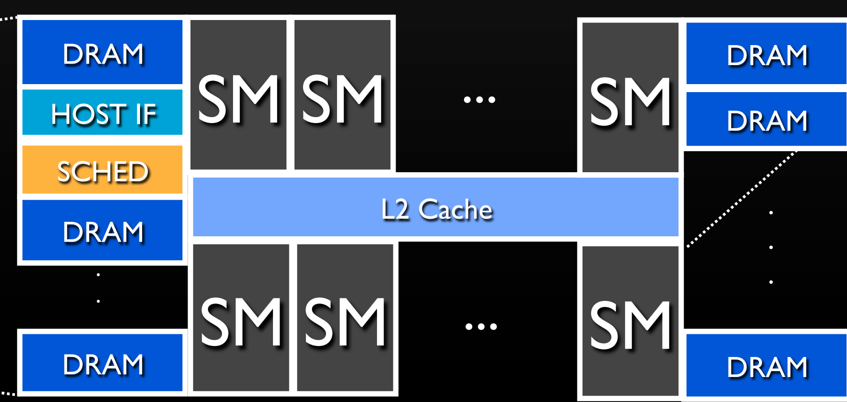


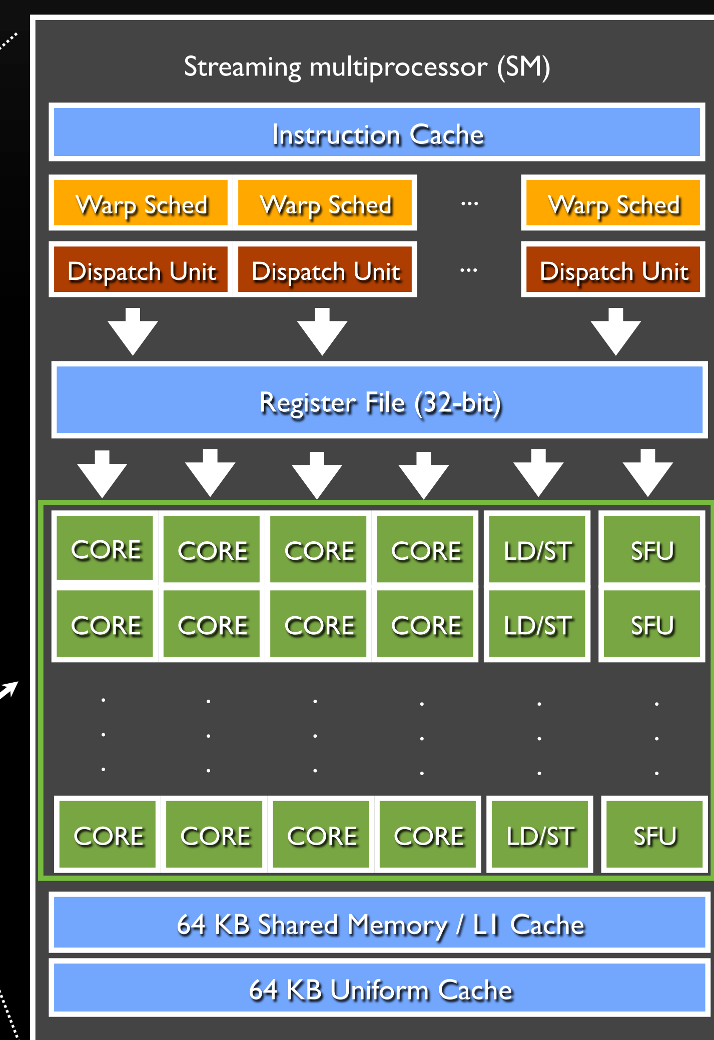
GPUs, NOT ONLY GAMING…

Massively parallel 32-bit many-core, GPGPU, transistors mainly used for arith
One integer or floating point instruction/clock cycle per thread/core

We usually run thousands of threads…







Kernel Function

Thread Block (up to 1024 threads)

Thread Warp (32 threads)

	NVIDIA FERMI (GTX 500 family)	NVIDIA KEPLER (GTX 700 family)
Cores	Up to 512	Up to 2880
SMs	Up to 16	Up to 48
Freq	Up to 1544 MHz	Up to 980 MHz
DRAM	Up to 3GB (192 GB/s)	Up to 6 GB (336 GB/s)

COFACTORING ON GPUs OUTLOOK

- Input: The set of candidate pairs (a,b) output by the sieve, the coefficients of the two polynomials
- Output: Indices of pairs (a,b) that are relations (or factors found)
- Two CUDA Kernels run sequentially:

1. Rational side: check $bf_r(a/b)$ for B_L-smoothness (discard bad)
2. Algebraic side: check $b^d f_a(a/b)$ for B_L-smoothness (output rels)

DESIGN STRATEGY

Each thread processes one or more pairs (a,b) (task parallelism!)

Each thread runs a fixed sequence of steps to determine if the polynomial value is B_r (B_a)-smooth except at most 3 (4) primes < B_L

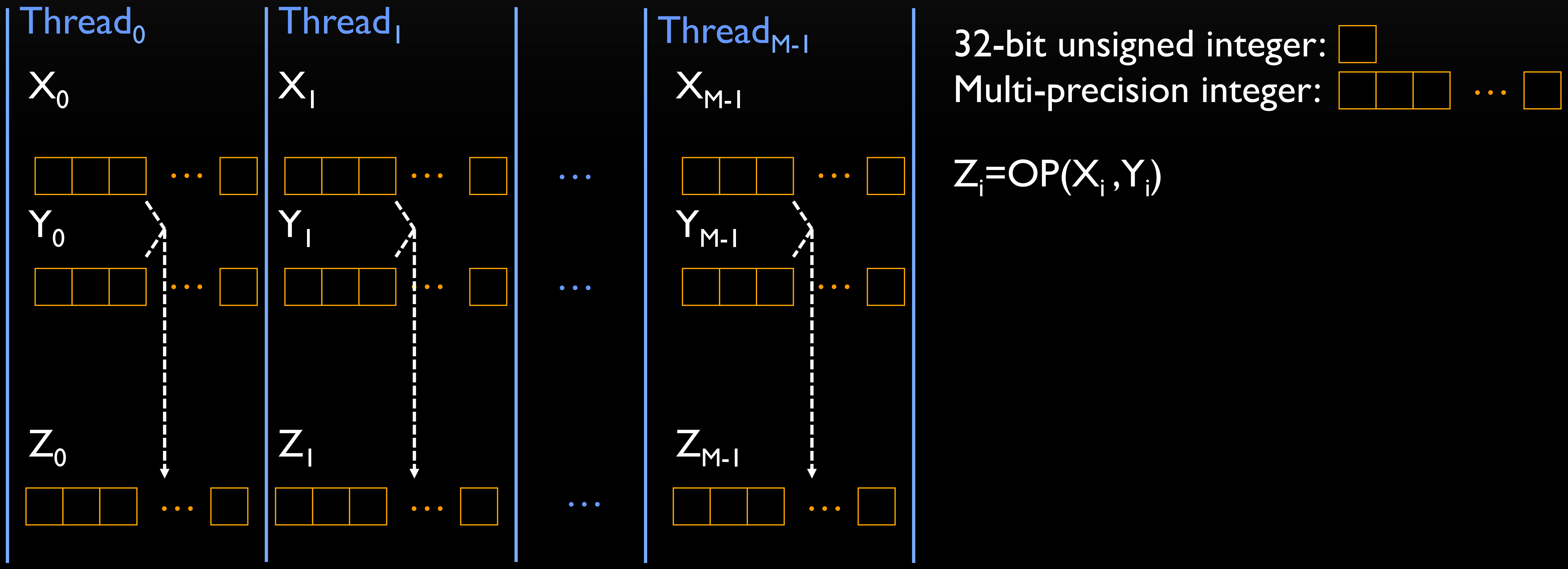
+ No thread synchronization, high computing/mem access ratio

- High register usage (and memory spilling…), high latency

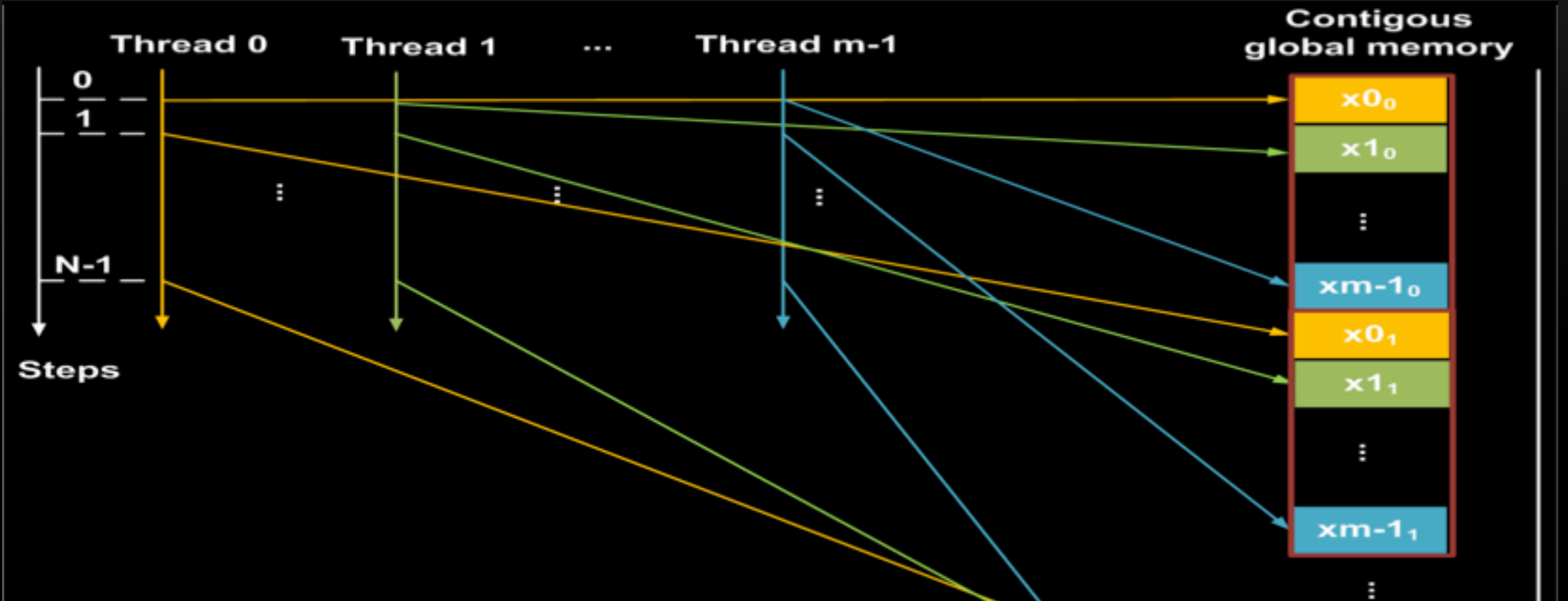
ARITHMETIC DESIGN

Sequential Radix 2^{32} Montgomery arithmetic

PTX level optimized code (heavy use of MAD instructions!)



GLOBAL MEM ACCESS: COALESCING



- Threads in a warp access adjacent memory addresses
- Single memory transaction for the whole block

KERNEL ANATOMY

PREAMBLE

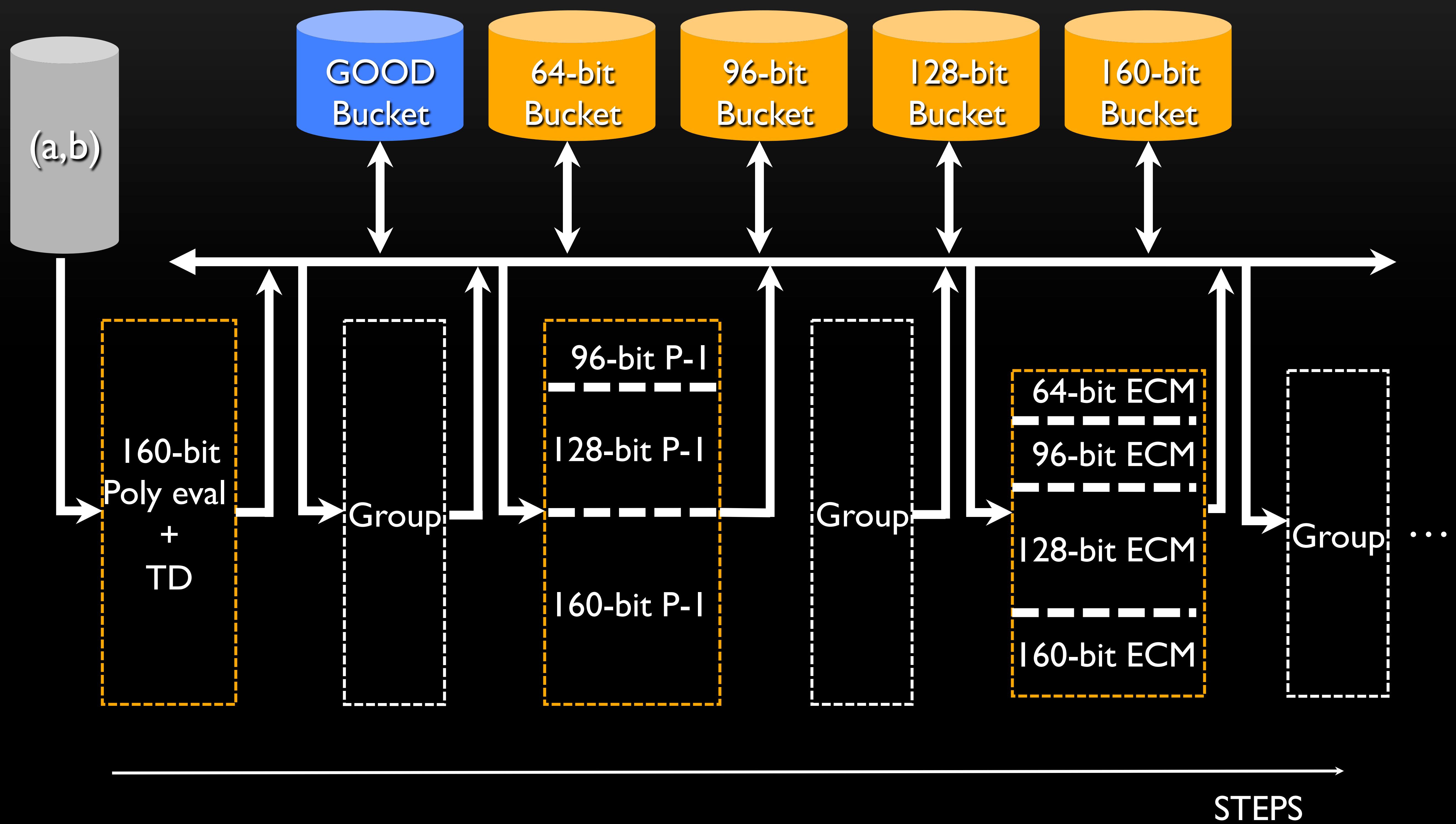
1. Read pair (a,b) from global memory and evaluate polynomial
2. Remove small factors: trial division

From now threads work on records: (value, index)

COFACTOR FACTORIZATION: REPEAT K TIMES

1. Group records in "buckets" according to #digits of value (distributed)
2. Factoring attempt: Pollard p-1 or ECM (unrolled code for bucket)
3. If factor found, divide out + pseudo primality (unrolled code for bucket)
4. Discard prime values > B_L (or cut-off), put aside smooth values $\leq B_L$

KERNEL WORKFLOW



ABOUT THE ALGORITHMS…

- Bivariate polynomial evaluation:
  naive, no Horner
- Trial Division:
  prime table in CMEM, divisibility test (Horner/Montgomery), exact div
- Pseudo primality test (Montgomery arithmetic):
  Selfridge-Rabin-Miller
- Pollard P-1 (Montgomery arithmetic):
  left-to-right modular exponentiation for stage 1, optimized BSGS stage 2
- ECM (Montgomery arithmetic):
  Twisted Edwards curves, add chains for stage 1[BK2012], opt. BSGS stage 2

INTEGRATION WITH RSA-768 SOFTWARE

Finding good parameters for GPU kernels is hard!

- Preliminary experiments: rule out bad configurations
- We have run many experiments on RSA-768 datasets

What to optimize for?

- We have fixed the yield, and looked for fastest configurations
- Focus on two cases: 95% and 99% yield
- Theoretical analysis confirmed experimental results

RSA 768: CHOICE OF PARAMETERS

- Values less than 2^{256} and $B_L=2^{37}$
- Trial division with 100-200 primes (algebraic side)
- One run of Pollard p-1: $B_1 \approx 2^{10}$, $B_2 \approx 2^{14}$
- 8-20 ECM runs: $B_1 = [2^8, 2^{10}]$, $B_2 = [2^{12}, 2^{15}]$

CPU vs GPU

CPU: INTEL I7-3770K 4 cores 3.5 GHz 16GB RAM

Large primes	Input pairs	Tot time	Sieve time	PS-cof time	Relations found
≤ 3	$\approx 5x10^5$	29.6s	25.6s	4.0s	125
≤ 4	$\approx 10^6$	32.0s	25.9s	6.1s	137

GPU: NVIDIA GTX 580 512 CORES 1544 MHz 1.5 GB RAM

Large primes	Input pairs	Desired yield	CPU/GPU Ratio	Time	Relations found
≤ 3	$\approx 5x10^5$	95%	9.8	2.6s	132
		99%	6.9	3.7s	136
≤ 4	$\approx 10^6$	95%	4.0	6.5s	159
		99%	2.7	9.6s	165

1CPU vs 1CPU + 1GPU

Large primes	# Input pairs	Setting	Total time	# Relations found	Relations/sec
≤ 3	$\approx 5\text{x}10^7$	No GPU	2961s	12523	4.23
		With GPU	2564s	13761	5.37
≤ 4	$\approx 5\text{x}10^7$	No GPU	1602s	6855	4.28
		With GPU	1300s	8302	6.39

Large primes $\leq$ 3: 24% GAIN
Large primes $\leq$ 4: 45% GAIN

CONCLUSIONS AND FUTURE WORK

- GPUs are a good accelerator for post sieving
- Their use can reduce overall NFS factoring time
- We will make the code available

- Optimize for NVIDIA Kepler GPUs (AMD?)
- Get actual figures for RSA 1024-bit