Cofactorization on GPUs

Andrea Miele[1], Joppe W. Bos[2], Thorsten Kleinjung[1], Arjen K. Lenstra[1]
[1]LACAL, EPFL, Lausanne, Switzerland
[2]NXP Semiconductors

NUMBER FIELD SIEVE

Asymptotically fastest known factoring algorithm.
Key to assessing security of RSA cryptosystem.
RSA 768-bit modulus factored with NFS in 2010.

Idea: to factor an odd composite n, find:
x,y: $x^2 = y^2 \pmod n$ and $x \neq \pm y \pmod n$

Two main phases:

- RELATION COLLECTION (90% OF TOTAL TIME)
- LINEAR ALGEBRA STEP (10% OF TOTAL TIME)

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) :

- $bf_r(a/b)$ is B_r-smooth except (at most) 4 large primes
- $b^df_a(a/b)$ is B_r-smooth except (at most) 4 large primes

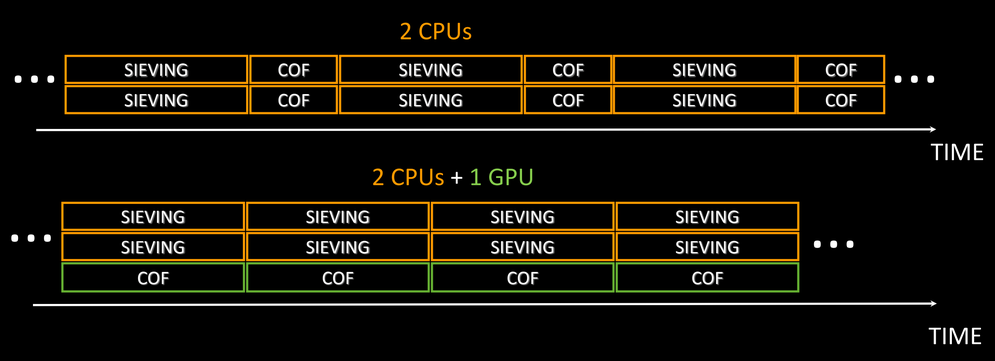
COLLECT RELATIONS (TASK PARALLELISM!)

SIEVING: find pairs (a,b): $bf_r(a/b)$ ($b^df_a(a/b)$) is product of a B_r – smooth (B_a– smooth) part and "small" cofactor

POST SIEVING (NORMALLY 12-17% OF THE TOTAL TIME):
1 Compute $bf_r(a/b)$ and $b^df_a(a/b)$ (200-400 bits)
2 Find small factors pair-by-pair (or re-sieve, bad on GPU)
3 Factor cofactors pair-by-pair (COFACTORIZATION)

FASTER NFS WITH GPUs?

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

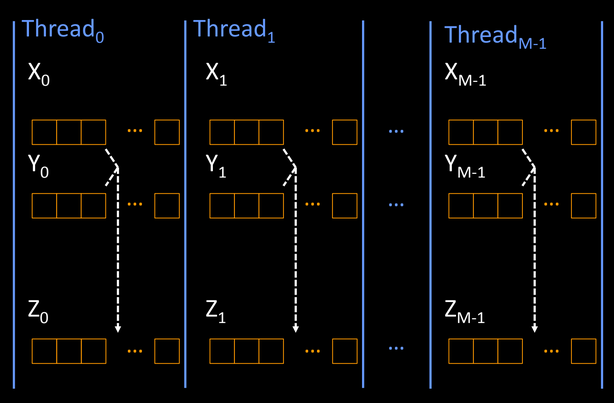


POST SIEVING ON GPUS: OUTLOOK

Input: Set of good candidate pairs (a,b) output by sieve
Output: The relations contained in the input set

Two Kernels run sequentially (one or more pairs per thread):
1.Rational side: check $bf_r(a/b)$ for smoothness (discard bad)
2.Algebraic side: check $b^df_a(a/b)$ for smoothness (output rels)

Sequential Radix 2^{32} Montgomery arith (coalesced GM access)
PTX optimized code (MAD, loop unrolling, SM as "scratchpad")



32-bit unsigned integer:
Multi-precision integer:

$Z_i = OP(X_i, Y_i)$

KERNEL ANATOMY

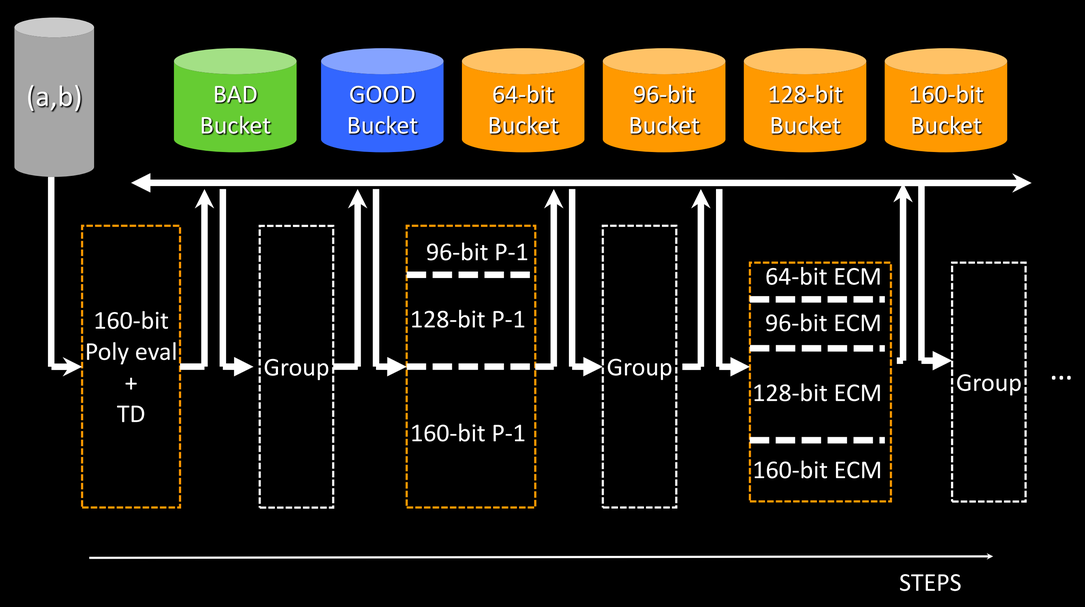
PREAMBLE

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

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

COFACTOR FACTORIZATION: REPEAT K TIMES (precomputed data in CMEM)

1. Group records in "buckets" according to val size (distributed)
2. Factoring attempt: Pollard p-1 or ECM (size specific code)
3. If factor found, divide + compositeness test (size specific code)
4. Discard prime values, put aside smooth values ("rels bucket")



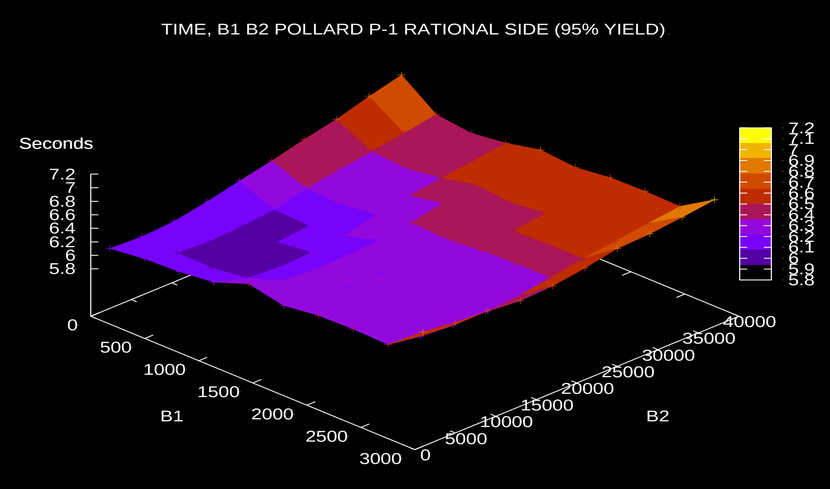
INTEGRATION WITH RSA-768 NFS SOFTWARE

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



CPU vs GPU

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

# Large primes	# Input pairs	Tot time	Sieve time	Cof time	#Rels
3	≈ $5x10^5$	29.6s	25.6s	4.0s	125
4	≈ 10^6	32.0s	25.9s	6.1s	137

NVIDIA GTX 580 512 CORES 1544 MHz 1.5 GB RAM

# Large primes	# Input pairs	Desired yield	CPU/GPU RATIO	Time	# Rels
3	≈$5x10^5$	95%	9.8	2.6s	132
		>99%	6.9	3.7s	136
4	≈10^6	95%	4.0	6.5s	159
		>99%	2.7	9.6s	165

1CPU vs 1CPU + 1GPU

# Large primes	# Input pairs	Setting	Tot time	# Rels found	Rels/sec
3	≈ $5x10^7$	No GPU	2961s	12523	4.23
		With GPU	2564s	13761	5.37
4	≈ $5x10^7$	No GPU	1602s	6855	4.28
		With GPU	1300s	8302	6.39

3 LARGE PRIMES: 27% GAIN
4 LARGE PRIMES: 50% GAIN

CONCLUSION AND FUTURE WORK

- Modern GPUs can accelerate NFS factoring
- Using GPUs can lead to lower factoring cost
- Optimize for Kepler GPUs
- Get figures for RSA 1024-bit