Parallel Sieve Processing on Vector Processor and GPU

Yasunori Ushiro (Earth Simulator Center)
Yoshinari Fukui (Earth Simulator Center)
Hidehiko Hasegawa (Univ. of Tsukuba)

Background

(1) RSA Cryptography is the key technology for safe Internet use.

(2) The safeness is based on the result that the factorization algorithm of a long-digit number n to P and Q has high computational complexity and consumes enormous computation time.

(3) To guarantee, a decryption time of more than 10 years is necessary, even using the fastest computer.

Another interests

- Different Architecture for RSA (GPU and Vector Processor instead of PC)
- Non-Floating Point Number Operations (Almost 0 GFLOPS)
- Other Usage of GPU and Vector Processor

RSA Cryptography

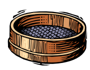
- Creation of keys
  (1) Choose prime numbers p, q, and e
  (2) Set n = p * q and f = (p-1) * (q-1)
  (3) Compute d = 1/e (mod f)
- Encryption
  Compute $C = M^{e}$ (mod n)
  Public keys (e, n) are used
- Decryption
  Compute $M = C^{d}$ (mod n); Euler's theorem $M^{f} \equiv 1$ (mod n)
  Secure keys (d, n) are used

Computation time of RSA-768 (232 digits)

	CPU･Year	ratio(%)
Sieve Processing	1500	90
0-1 Matrices Proc.	155	9
Exploration of polynomial	20	1
Algebraic Square root	1	0
Others	1	0

Cf. AMD64 (2.2GHz, 1 Core)
Dimension: 192,796,550 * 192,795,550

Sieve method



Factorize N based on the relationship:
$A^2-B^2=(A-B)(A+B)\equiv 0$ (mod N)

(1) Gather **numerous** A_i and B_i such that $A_1^{l1}\cdot A_2^{l2}\cdots A_k^{lk}\equiv B_1^{m1}\cdot B_2^{m2}\cdots B_j^{mj}$ (mod N)

(2) Look for even l_i and m_i with factorization of 0-1 Matrices

Steps of Sieve Processing

		Comp.	Iteration	Size
1	Set Number	Long	1	10^{200}
2	Compute Base	int	1	10^8
3	Choose prime number Q_i and $f_i(x)$	int	1	10^3
4	Process for each I		10^3	
4.1	Repeat Step 4.1.1		(10^{13})/LP	
4.1.1	Compute PS[i] V[i]=0 Sieve Processing	Double int int64	LP LP N*LP/p	p (Harmonic mean of primes)

LP: 10^6 for PC, 10^8 for GPU, 10^9 for ES2

	1	2	3	4	5	6	7	8	9	10	11	12	13	…	LP
Initialize	1	1	1	1	1	1	1	1	1	1	1	1	1	…	1

	1	2	3	4	5	6	7	8	9	10	11	12	13	…	LP
Initialize	1	1	1	1	1	1	1	1	1	1	1	1	1	…	
2		2		2		2		2		2		2		…	

	1	2	3	4	5	6	7	8	9	10	11	12	13	…	LP
Initialize	1	2	1	2	1	2	1	2	1	2	1	2		…	
2		2		2		2		2		2		2		…	
3			3			3			3			3		…	

	1	2	3	4	5	6	7	8	9	10	11	12	13	…	LP
Initialize	1	2	3	2	1	6	1	2	3	1	1	6	1	…	
2		2		2		2		2		2		2		…	
3			3			3			3			3		…	
5					5					5				…	

	1	2	3	4	5	6	7	8	9	10	11	12	13	…	LP
Initialize	1	2	3	2	5	6	1	2	3	10	1	6	1	…	
2		2		2		2		2		2		2		…	
3			3			3			3			3		…	
5					5					5				…	
p	p					2p					3p			…	

N primes ← p → **LP/p** Steps

4.1.1 Kernel of Sieve

```
for (k=0; k<N; k++) : # of primes in base
  { for (i=Start[k]; i<LP; i+=Prime[k])
     { V[i] += LogP[k]; }   LP : Size of Sieve
  }
for (i=0; i<LP; i++) : Pick up sieved data
  { if(V[i] >= PS[i]) { Sive[No] = Pointer + i;
                        No++; }
  }                      No : # of sieved data
Update Start[0]～Start[N-1] for next Sieve
```

13

Tuning for Sieve Processing

Base must be stored in fast memory

- PC (Cache)
  Shorter LP (LP is size of Sieve) 10^6
- Vector Processor ES2
  LP=10^9; 10^3 times larger than that of PC
  Picking up of sieved data is slow (“**if**” exists in a loop)
- GPU(GTX480)
  LP=10^8; 10^2 times larger than that of PC
  Discontinuous memory access (stride varies)

14

Tuning for Vector Processor ES2

- Picking up ratio is 10^{-6} ～10^{-10}
- Compute Maximum value in a block instead of picking up
- Modification:
  If the maximum number > value_B
  then perform Picking_up process
- Block size is 64K (65,536)
- This results in 3 times faster

15

Tuning for GPU

- Each thread has smaller LP
  (LP of PC * 1000 / 20480 threads)
- Larger Prime[k] makes small hit ratio
  if(ii < LP) V[ii] += LogP[k];
- N is used for Loop length instead LP
  → Incorrect result for V[ii]+=LogP[k];
- Sieve Processing can permit small error
  (Loss of pick up is OK up to 10^{-3})
  → 3 types of parallelization are used based on the size of prime numbers in the base

16

GPU program (before)

```
no = gn*bn;                       bn=512, gn=40
for (k=0; k<LP; k+= no)           LP=250*1024^2
  { i = (bn * blockIdx.x + threadIdx.x) + k;
    V[i] = 0; }
__syncthreads();
for (k=0; k<N; k++)               20,480 threads in LP
  { for (i=Start[k]; i<LP; i+=Prime[k]*no)
     { ii = (bn*blockIdx.x+threadIdx.x)*Prime[k]+i;
       if(ii < LP) V[ii] += LogP[k]; }
    __syncthreads(); }
```

17

GPU program (after)

```
for (k=0; k<N1; k++)              Sieve for 1～5120(5K)-th primes
  { for (i=Start[k]; i<LP; i+=Prime[k]*gn*bn)
     { ii = (bn*blockIdx.x + threadIdx.x)*Prime[k] + i;
       if(ii < LP) V[ii] += LogP[k];   __syncthreads(); }  }
for (k=N1; k<N2; k+=gn)           5K+1～40K-th primes
  { kk = blockIdx.x + k;
    for (i=Start[kk]; i<LP; i+=Prime[kk]*bn)
     { ii = threadIdx.x*Prime[k] + i;
       if(ii < LP) V[ii] += LogP[kk];   __syncthreads(); }  }
for (k=N2; k<N; k+=gn*bn)         Over (40K+1)-th primes
  { kk =(bn*blockIdx.x + threadIdx.x) + k;
    for (i=Start[kk]; i<LP; i+=Prime[kk])
     { if(i < LP) V[i] += LogP[kk];   __syncthreads(); }  }
```

18

Change of Result on Modified Program

- 60 digits, LP=$250*1024^2$, Prime numbers N=10,000*1024, 5000 iterations
- M: Number of Sieved data
  - (1) Original: M=448425
  - (2) After parallelization (12times)
    - M=[448338, 448367], mean=448350
    - largest difference = 87 (0.02%)

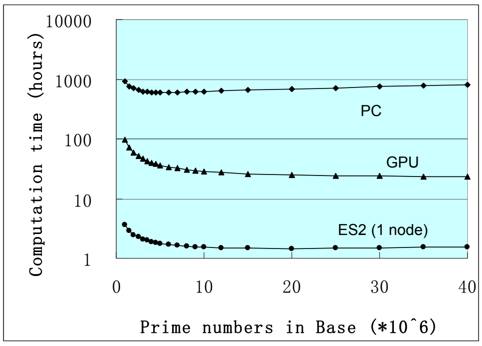
19

Conditions

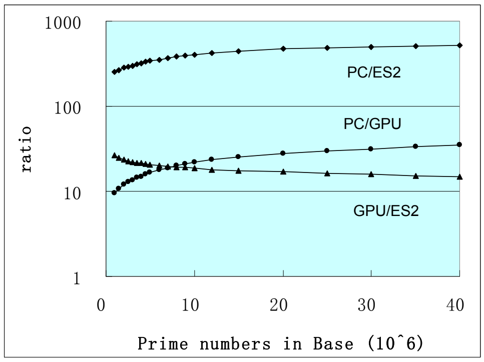
- PC: 1 Core of Dell Vostro 200
  - Intel Core 2, 2.33GHz, 2GB
  - Windows Vista, gcc, -O3
- Vector Processor ES2 1 node (8 CPU)
  - 3.2GHz：819Gflops, 128GB
  - SUPER-UX, Auto-parallel FORTRAN+MPI
- GPU
  - NVIDIA GTX580 1.544GHz, 1.5GB
  - Unix, CUDA 3.2, -O3

20

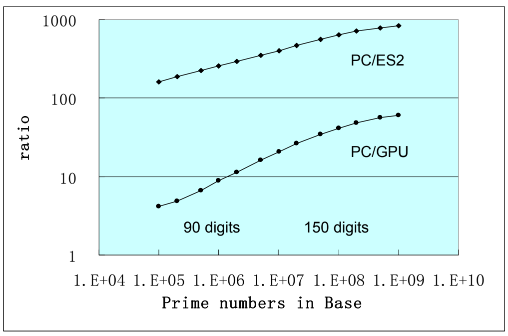
Time of Sieve Processing 60 digits



Ratio of Sieve Processing 60 digits



Dependency of Base size



ES2 (Vector Processor)

Parallelization is simple and easy, however a special treatment is needed for storing sieved data.

GPU

Fine grain Parallelization is needed, and that makes data dependency for sieved data. By omitting some data inconsistencies, we chose “forced parallelization”.

Summary of Sieve Processing

- Almost all ops. are addition of 32 bits integer with different-stride
- 99.9 % are vectorizable, and easy MPI-parallelizable
- Speed depends on the size of fast memory
- One node of ES2 is 200 – 800 times faster than PC (Check before pick up becomes 3 times faster)
- GPU (GXT580) is 5 – 60 times faster than PC (Force-parallelization has 0.02% loss of sieved data; 3 types of parallelization are used based on the magnitude of primes)
- High speed range of Base is ES2 >> GPU >> PC

Forecast of RSA-768 (232 digits)

	ratio (%)	Time(Node・Year)		
		CPU	GPU	ES2
Sieve Processing	90	1500	25	2
0-1 Matrices	9	155	4	0.3
polynomials	1	20	0.5	0
Alg. SQRT	0	1	0	0
Others	0	1	0	0

Prediction (GPU, ES2)

Guess of RSA-1024 (309 digits)

	ratio (%)	Time(Node・Year)		
		CPU	GPU	ES2
Sieve Processing	81	$6*10^5$	10^4	750
0-1 Matrices	19	$14*10^4$	3500	280
Polynomials	0	2000	50	4
Alg. SQRT	0	100	3	0
Others	0	100	3	0

Guess is based on Sieve 20^2, 0-1 Mat. 30^2, Others 10^2