Designing a Multiprecision Number Theory Library

C. Hrițcu **I. Goriac** **R. M. Gordân** **E. Erbiceanu**

Faculty of Computer Science
“Al. I. Cuza” University of Iaşi
6600 Iaşi, România
E-mail: {donald,gi,grm,shiana}@infoiasi.ro

Why multiprecision computing?

- Security protocols for Internet applications
  - e-commerce
  - e-payment
  - e-auction
- Cryptography
  - Public key encryption
  - Digital signatures
  - Hash functions
- Many other domains
  - Numerical calculus
  - Probabilities and statistics

Available solutions

- *C* and *C++* have **limited precision**
  - A "*long*" has usually only 32 bits
  - A "*long long*" (*gcc*) typically has 64 bits
  - A "*long double*" uses
    - 52 bits for the mantissa
    - 11 bits for the exponent
- *Java* has multiprecision capabilities
  - **Highly portable**
  - **Not** so **efficient**

Alternatives

- Programs like *Maple* or *Mathematica*
  - Unlimited precision
  - Easily prototype algorithms
  - Easily compute constants
  - **Not efficient**
  - **Not portable**
- Multiprecision libraries
  - **Most efficient** solution
  - Many of them are free software (GNU GPL)
    - LIP, LiDIA, CLN, NTL, PARI, GMP, MpNT etc.

LIP (Large Integer Package)

- Written by Arjen K. Lenstra and later maintained by Paul Leyland
- One of the first
- ANSI C
- Highly portable
- Not efficient

LiDIA (a library for computational number theory)

- Developed at the Technical University of Darmstadt (Thomas Papanikolau)
- C++ library
- Highly optimized implementations
  - Multiprecision data types
  - Time-intensive algorithms
- Can use different integer packages (like Berkley MP, GMP, CLN, libI, LIP etc.)

CLN (a Class Library for Numbers)

- Written by Bruno Haible and currently maintained by Richard Kreckel
- C++ library that implements elementary arithmetical, logical and transcendental functions
- Rich set of classes
  - Integers
  - Rational numbers
  - Floating-point numbers
  - Complex numbers
  - Modular integers
  - Univariate polynomials etc.
- Memory and speed efficient

NTL (a Library for doing Number Theory)

- Written and maintained mainly by Victor Shoup
- C++ library
- High performance
  - Polynomial arithmetic
  - Lattice reduction
- Portable
- Can be used in conjunction with GMP for enhanced performance

PARI/GP

- Developed at Bordeaux by a team led by Henri Cohen
- Formal computations on recursive types at high speed
- Primarily aimed at number theorists
- Extensive algebraic number theory module
- Can be used as a calculator (GP)

GMP (GNU Multiple Precision arithmetic library)

- Developed by Törbjord Granlund and the GNU free software group
- C library for arbitrary precision arithmetic
- General **emphasis on speed**
- Highly optimized ASM
  - for the most common inner loops
  - for a lot of CPUs
- **Faster** than most multiprecision libraries
- Its advantage **increases** with the operand sizes

MpNT (a Multi-Precision Number Theory package)

- Developed at the Faculty of Computer Science, "Al. I. Cuza" University of Iaşi
- ISO C++ library
- Cryptographic applications
- Integer and modular arithmetic (for now)
- Characteristics
  - Speed efficient
  - Highly portable
  - Code structure and clarity were not disregarded
- Free software (GNU Lesser GPL)

Basic Principles

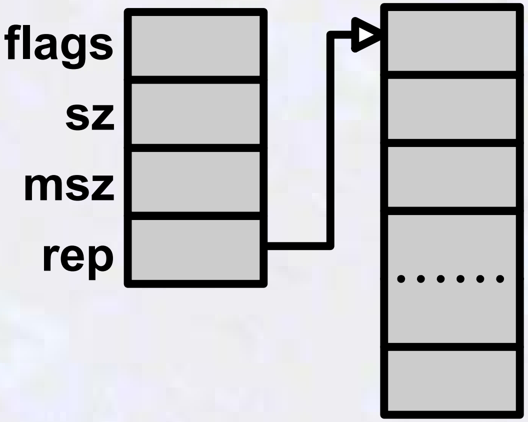
- Goals
  - Efficiency
  - Portability
  - Functionality
- Choices and tradeoffs
- Many products
- The user has the opportunity to choose
- Certain common lines should be followed while designing multiprecision number theory library

Programming Language

- Low-level (assembly)
  - Fastest
  - Not portable
  - Very hard to maintain
- High-level
  - Easily portable
  - Easy to understand and maintain
  - Some efficiency loss
- The compromise solution is to use both
- C++ is probably the best for multiprecision computing
- **MpNT** uses ISO C++ for the main part of the library
- ASM is used only for the most frequently called functions forming a a small machine-dependent kernel.

Number representation

- Depends on what the hardware provides
  - Registers dimensions
  - Instructions set
  - Cache sizes
  - Parallelism level
- **MpNT** uses signed-magnitude representation for its multiprecision integers (*MpInt* class)
- The current implementation includes
  - flags
  - size
  - allocated size
  - pointer to representation
- Quick access to members
- Easily extendible



Library Structure

- The best approach is to group the functions in layers
  - Only low-level functions have direct access to number representation
  - High-level ones have a higher degree of independence
- **MpNT** has two layers
  - The kernel
  - The C++ classes

MpNT Kernel

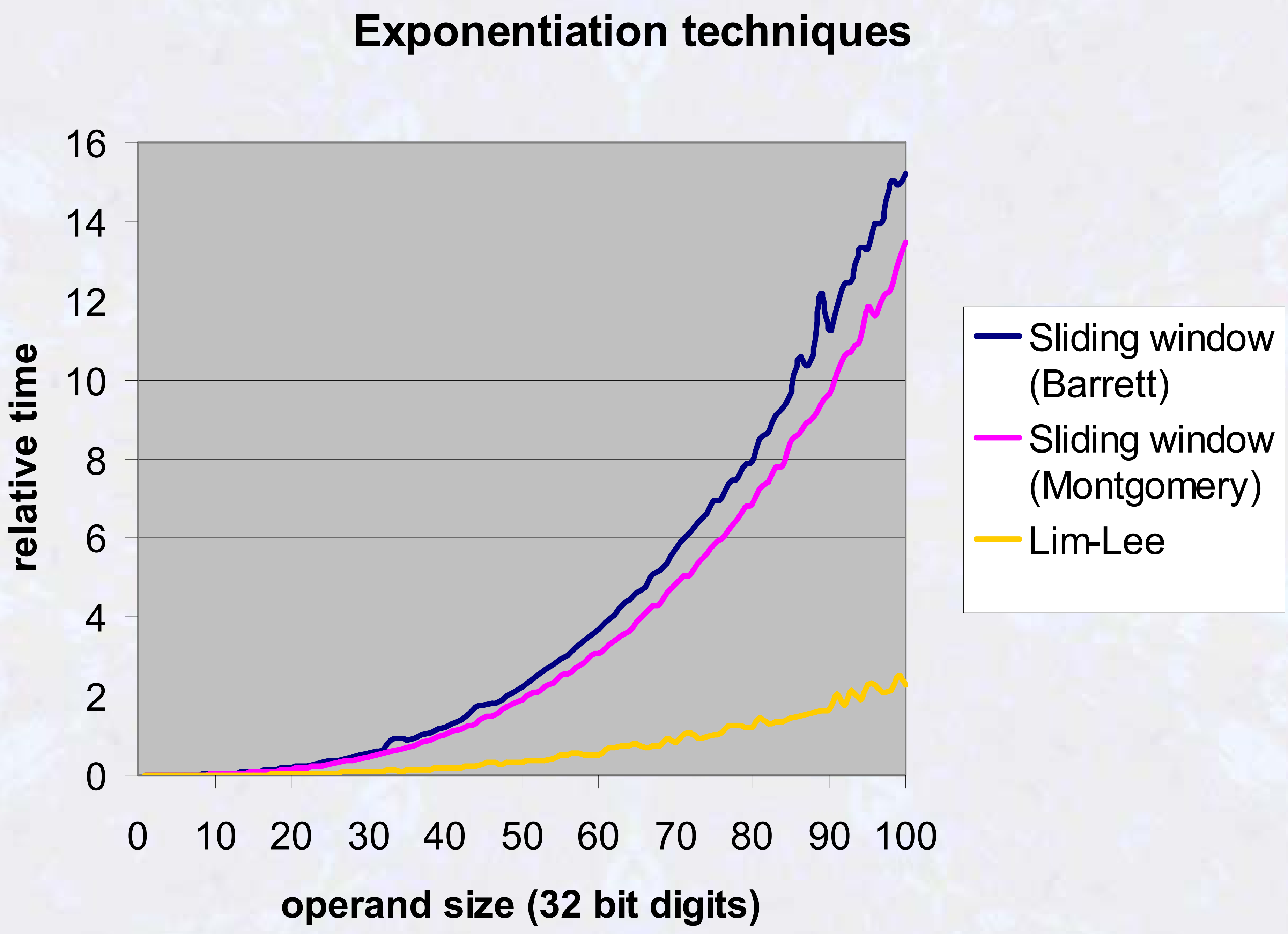
- Contains small routines
  - carefully optimized
  - easy to rewrite
- Most of the functions operate on unsigned arrays of digits:
  - Comparisons
  - Bitwise operations
  - Basic arithmetical operations
- Dangerous to call directly
- Optimizations apply for the Intel IA-32 CPUs
- MpNT might use the GMP or CLN kernel

MpNT C++ Classes

- MpInt
  - Multiprecision integer arithmetic
  - Basic arithmetical operations (and **more**)
  - All the operators available for *int* are overloaded
  - Hides the functions of the kernel
- MpMod
  - Multiprecision modular arithmetic
  - One modulus can be used at any time
  - The numbers are always modularly reduced
  - Basic modular operations
  - High performance modular reduction, multiplication and exponentiation – using pre-computed modulus information (classes ***MpModulus*** and ***MpLimLee***)

Algorithm selection

- In many cases several algorithms may be used to perform the same operation
  - The ones with the best *O*-complexity are preferred when dealing with **huge** numbers
  - On smaller numbers simpler, more optimized algorithms may perform much better
- Performance testing is required to find the limits of applicability
- In **MpNT** we implemented a lot of algorithms but the interface will use only the routines (or combination) that proved to be most efficient


Exponentiation techniques
relative time
16
14
12
10
8
6
4
2
0
0
10
20
30
40
50
60
70
80
90
100
operand size (32 bit digits)
Sliding window (Barrett)
Sliding window (Montgomery)
Lim-Lee

Memory Management

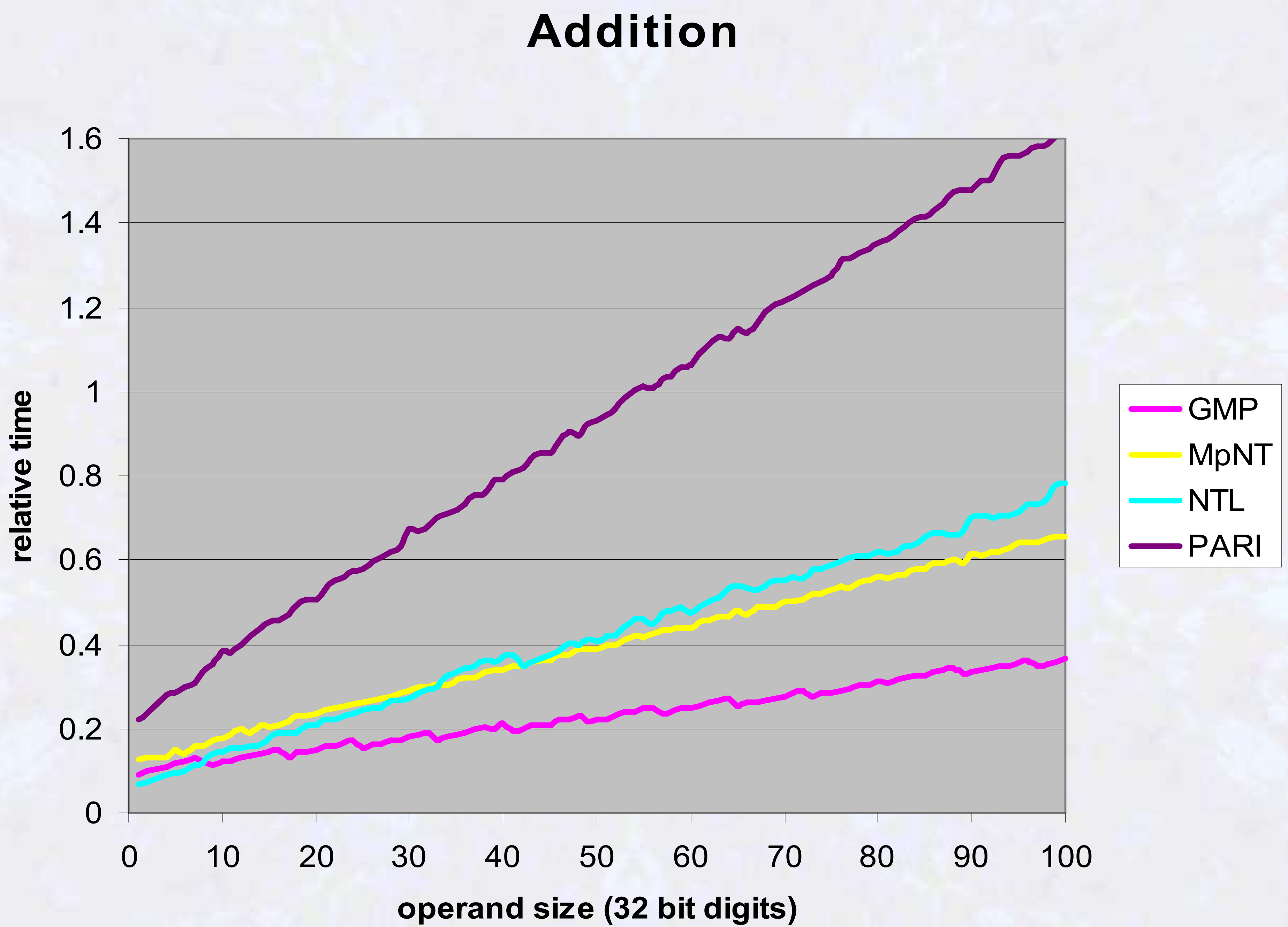
- Most frequently on demand allocation
- Space may be transparently allocated whenever a variable needs it
- Memory leaks may be prevented by the use of class destructors
- Some libraries
  - Offer garbage collection (e.g. CLN)
  - Allow the user to chose the memory management policy (e.g. LiDIA)
- **MpNT** uses explicit allocation of memory
  - Frequent reallocation is avoided

Error Handling

- It is desirable to signal the occurred errors, but…
  - This is time consuming
  - Makes code harder to read and maintain
- A frequent approach is to ignore errors
  - Involves some risks
  - Eliminates the overhead
- We chose not to ignore errors
  - **MpNT** uses the throw-try-catch mechanism provided by C++ to signal errors to the user

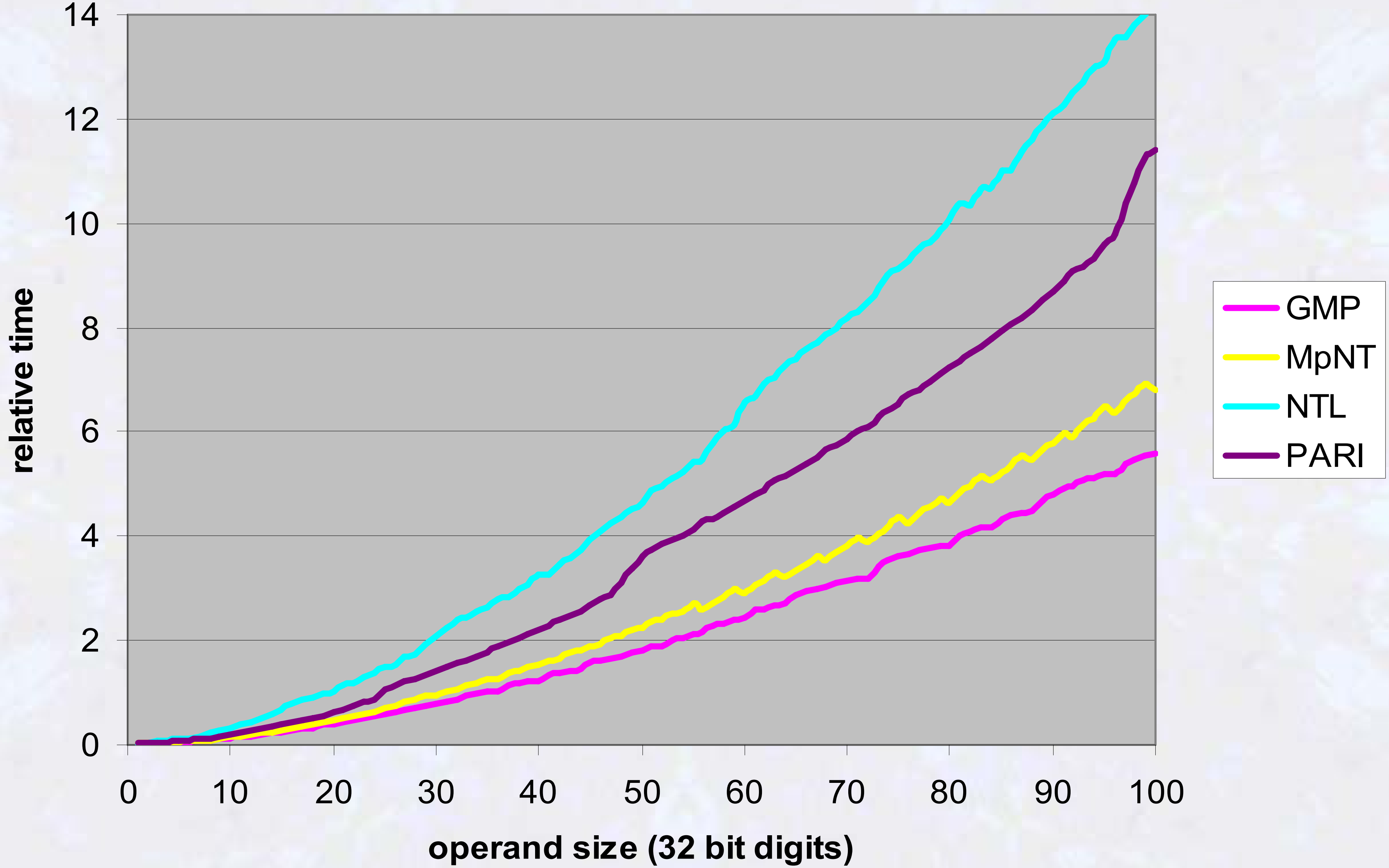
Comparisons

- Library versions
  - CLN 1.1.5
  - GMP 4.1
  - MpNT 0.1 pre-release
  - PARI 2.2.4 alpha
- Test system
  - AMD K6 800MHz
  - 256MB RAM
  - Mandrake Linux 9.0

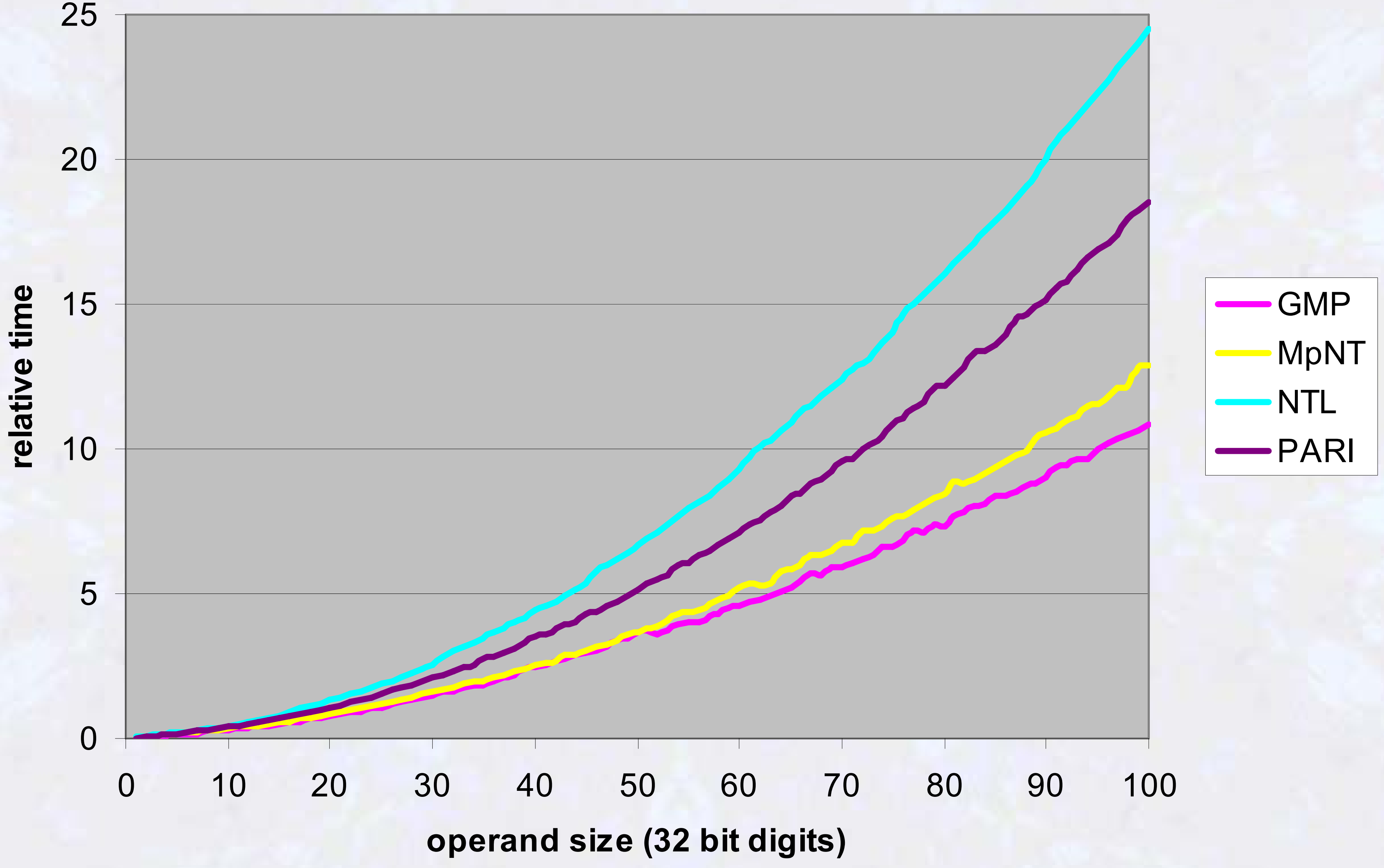


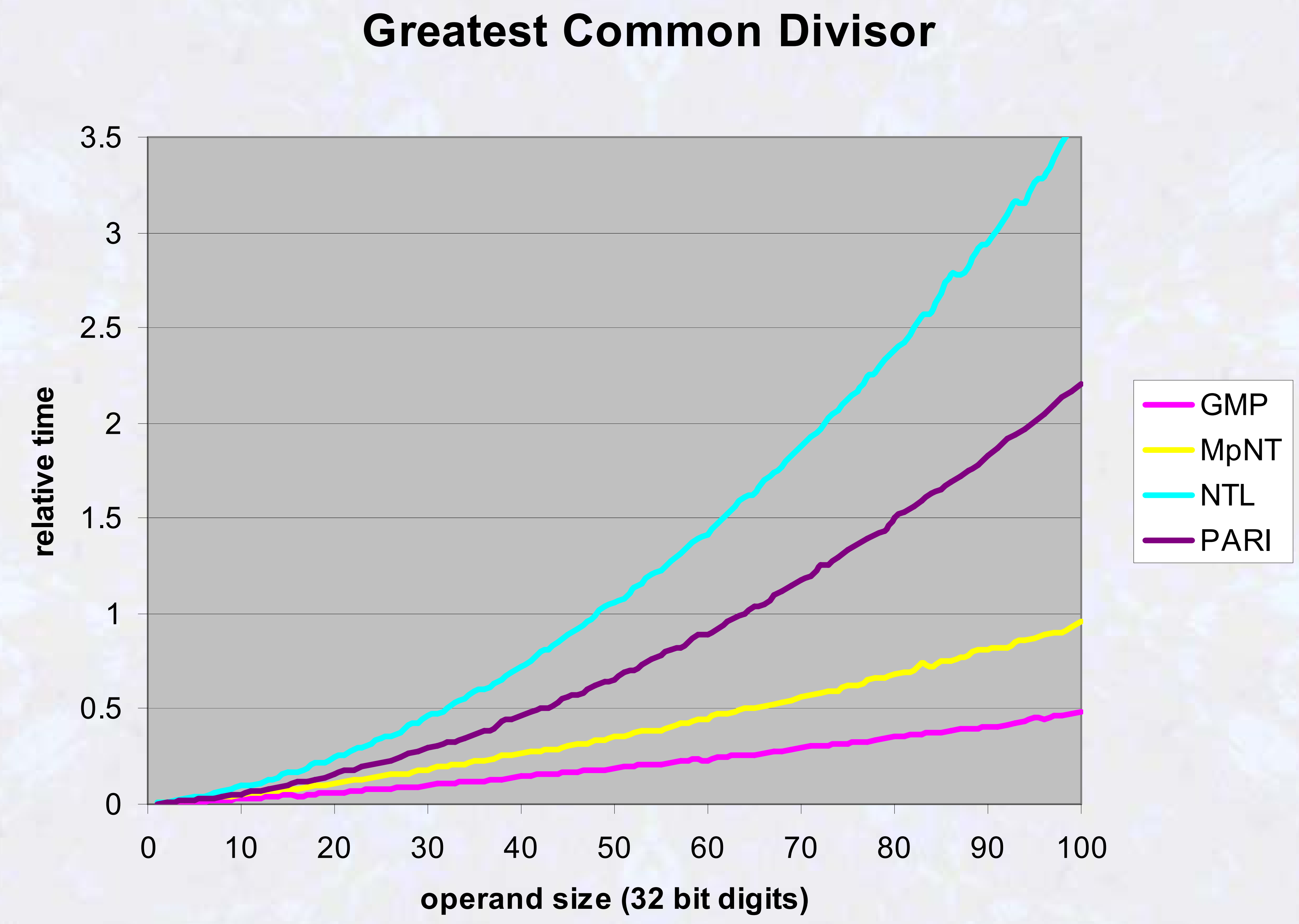
Addition
relative time
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
10
20
30
40
50
60
70
80
90
100
operand size (32 bit digits)
GMP
MpNT
NTL
PARI

Multiplication

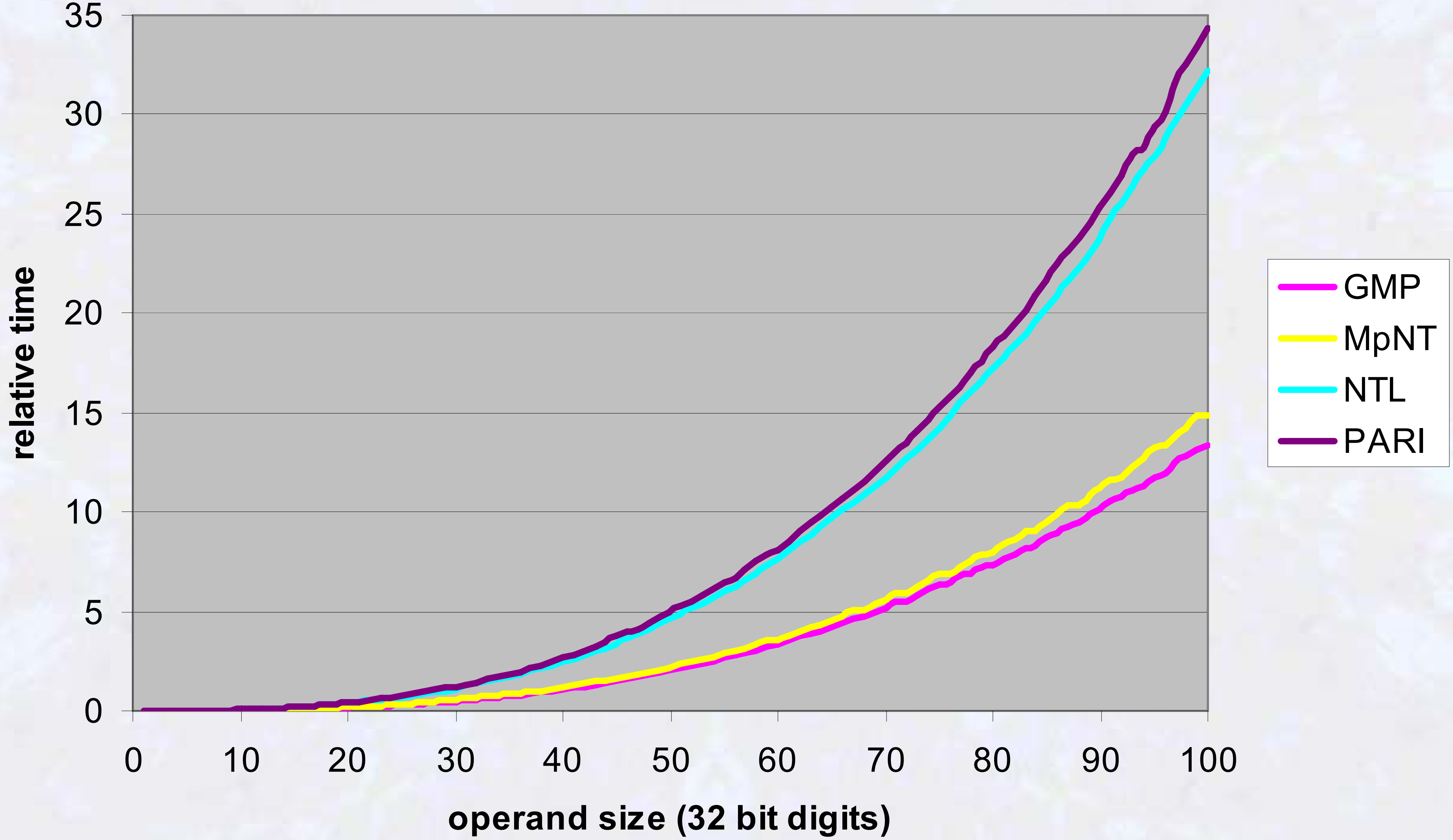


Modular reduction

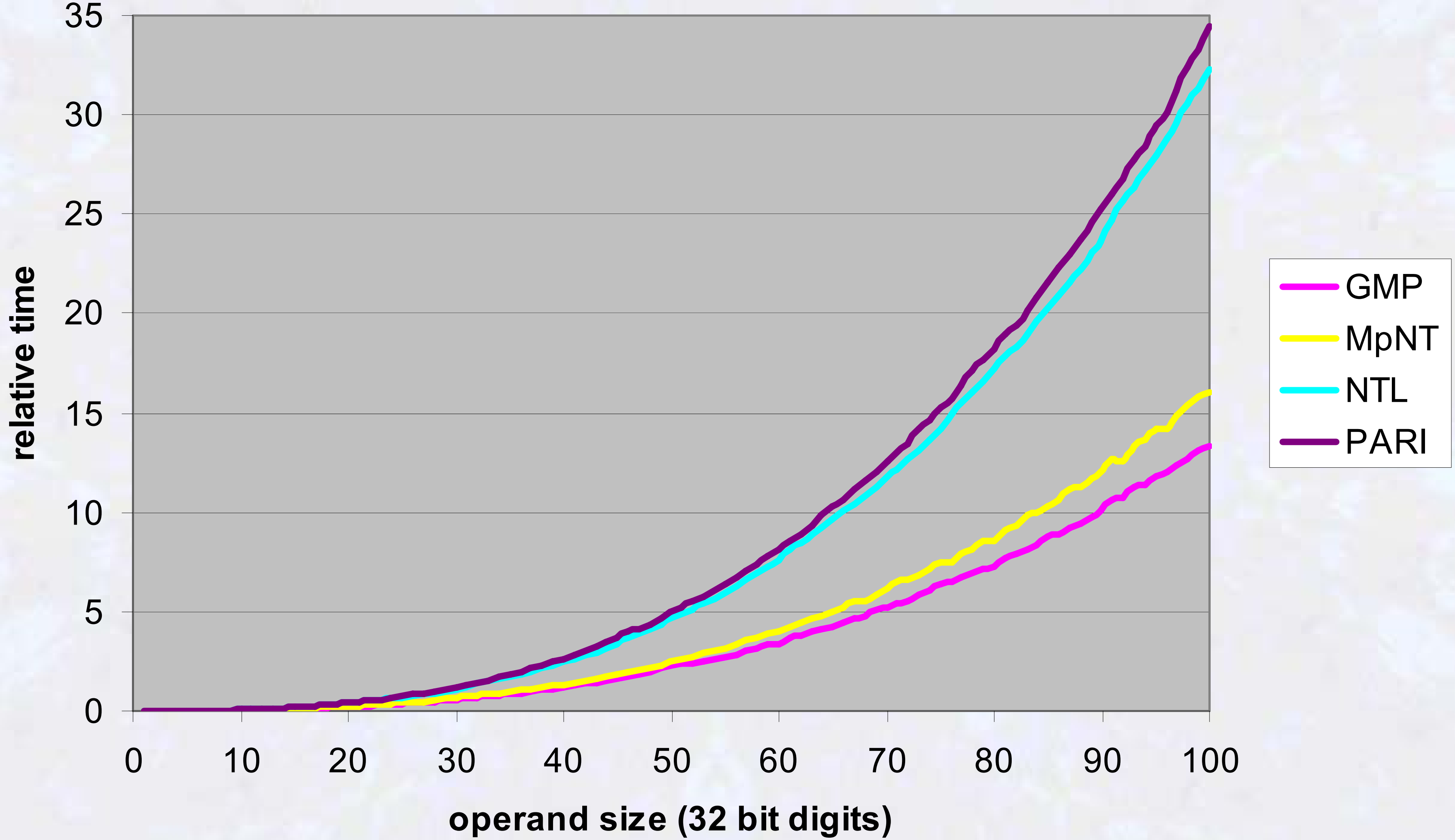



Greatest Common Divisor
relative time
3.5
3
2.5
2
1.5
1
0.5
0
0
10
20
30
40
50
60
70
80
90
100
operand size (32 bit digits)
GMP
MpNT
NTL
PARI

Modular exponentiation (odd modulus)



Modular exponentiation (even modulus)



Questions?

- Thank you