大整数 Comba 和 Karatsuba 乘法的多核并行化研究①

蒋丽娟, 刘芳芳, 赵玉文, 杨 超, 蔡 颖

(中国科学院 软件研究所, 北京 100190)

摘 要: 大整数运算广泛地应用于公钥加密算法、大规模科学计算中高精度浮点数运算类以及构建大特征值等领域, 然而其大部分算法空间和时间开销都很大, 尤其对于核心运算之一的大整数乘法, 当数据达到一定规模时, 超长的串行计算时间已成为制约算法应用的巨大瓶颈. 近几年来, 伴随着多核、众核芯片的迅猛发展, 通过充分挖掘算法本身的并行度以利用并行处理器的强大计算能力, 进而高效地提升算法性能, 成为一种研究趋势. 本文基于通用多核并行计算平台, 研究了大整数乘法 Comba 及 Karatsuba 快速算法的并行化, 提出了高效的多核并行算法. 在算法实现及性能优化上, 采用了 OpenMP+SIMD 的多级并行技术, 使性能获得巨大提升. 在性能测试上, 我们使用优化的并行算法与原始串行算法进行对比试验, 结果显示, 8 线程并行 Comba 算法和 Karatsuba 算法相比串行对应算法分别实现了 5.85 倍以及 6.14 倍的性能加速比提升.

关键词: 大整数运算; Comba 算法; Karatsuba 算法; OpenMP; SIMD

Multi-Core Parallel of Large Integer Multiplication Comba and Karatsuba Algorithms

JIANG Li-Juan, LIU Fang-Fang, ZHAO Yu-Wen, YANG Chao, CAI Ying

(Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: The operations of large integers have been widely used among the fields of public-key encryption algorithms, the operations of floating-point data types of large-scale scientific computation and the construction of large eigenvalues and so on. However, most of the large integer arithmetic algorithms are space and time consuming especially for the large integer multiplication, one of the core large integer operations. When data reaches a certain scale, the overlong serial computing time has been the bottleneck of the applications of the large integer algorithms. Simultaneously, with the popularity of multi-core processors in the computer field in recent years, taking advantages of the parallelism of algorithms, it'll be a trend to parallelize applications to optimize their performance efficiently by using multithread programming to take full use of the powerful computing capability of parallel computers. Meanwhile, the paper did an in-depth study on the multi-core parallelization of fast algorithms of large integer multiplication Comba algorithm and Karatsuba algorithm using the OpenMP multithread programming technology and automatic vectorization technology about the SIMD model of the Intel C++ Compiler. Besides, the testing shows that the speedup of Comba algorithm with 8 threads reaches to 5.85, and Karatsuba algorithm with 8 threads reaches 6.14 at most.

Key words: large integer arithmetic; comba algorithm; Karatsuba algorithm; OpenMP; SIMD

大整数乘法作为大整数运算的核心计算之一, 应用非常广泛, 大多公钥加密算法如 RSA、ECC、Diffie-Hellman 等算法中, 都要用到大整数乘法[1]. 长久以来, 人们一直致力于大整数乘法算法的研究, 传统的基线乘法, Comba 算法, Karatsuba 乘法, Toom 乘法, FFT 算法, 以及 Schönhage-Strassen 等算法的逐步被发掘, 使得大整数乘法在运算规模以及性能提升方面研究成果显著.

一方面, 目前市场上多核处理器已成为主流, 通过重新设计程序来充分利用多个核的硬件资源, 可以

① 收稿时间:2016-02-25;收到修改稿时间:2016-03-24 [doi: 10.15888/j.cnki.csa.005408]

© 中国科学院软件研究所 http://www.c-s-a.org.cn

显著提升程序性能. 目前基于多核处理器进行并行的技术主要包括基于共享内存模型的 OpenMP 多线程编程技术以及 Pthread 技术等, 而 OpenMP 技术被大部分较为流行的编译器所支持(如 Intel C++ compiler, GCC Compiler 等)[2], 同时数据并行 SIMD 向量化技术及其自动化在 Intel 的 Core 微架构也已较为成熟, 软件的并行化已是大势所趋. 另一方面, 大整数乘法运行时间及其所消耗的资源随着数据规模的加大将大幅增加, 大整数乘法运算性能的提升一直是国内外研究热点. 利用多核处理器强大的并行计算资源, 采用并行化方法来提升大整数乘法性能是一种有效途径.

Comba 以及 Karatsuba 算法作为重要的大整数乘法底层算法, 如果能够提高其性能, 在实际应用中将有着十分重要的作用. Comba 算法并行优化方面, Vladislav Kovtun 和 Andrew Okhrimenko 曾采用延迟进位技巧改进 Comba 算法[3], 并采用 section 策略和 for 策略两种并行方案优化算法, 与串行算法相比, 可分别达到 1.5 倍及 2 倍加速[2], 该实现虽已达到较好的加速, 但 section 策略并行度低, 而 for 策略采用静态数据分配容易造成负载不均衡, 影响算法性能提升; Karatsuba 算法并行优化方面, Tudor Jebelean 曾在 9 核 paclib 计算机中将算法进行 3 线程和 9 线程并行, 其中基为 2^{29}, 与串行算法相比, 3 线程并行最高可达 2.93 倍加速, 9 线程最高可达 7.98 倍加速[4], 有较好的性能加速, 但依赖于 3 核平台或 9 核平台, 数据表示能力有限.

本文将基于 OpenMP 技术及 SIMD 自动向量化技术, 研究大整数乘法 Comba 及 Karatsuba 快速算法在 8 核 Intel Xeon 55 系列处理器平台上的多核并行化.

1　Comba算法及其并行化

Comba 算法基于基线乘法, 基线乘法进行运算时将乘数与被乘数的每一位进行相乘, 每相乘一次, 便实施进位操作; 而 Comba 算法可首先通过多次乘加操作得到中间结果的一个数位, 即中间结果的每个数位是乘数与被乘数的多对对应位相乘后相加的结果, 计算得到中间结果的数位后, 向高位进位, 按照同样的方法计算剩余数位, 从而顺序求得结果.

1.1 Comba 算法描述

设被乘数 u、乘数 v, 位数分别为 un 和 vn, $u_{iu}(0 \le iu < un)$ 和 $v_{iv}(0 \le iv < vn)$ 分别表示被乘数和乘数的数位, 则乘积 r 的数位 $r_{ir}(0 \le ir \le un+vn)$ 的计算可按照如下公式(1)计算:

$$r_{ir} = \sum_{ir=iu+iv} u_{iu} * v_{iv} \tag{1}$$

算法根据公式(1)循环获得乘积的某一位 $r_{ir}(0 \le ir \le un+vn)$, 实施进位, 从而顺序求得最终结果[5]; 循环由乘积 r 的位数控制, 每次循环乘数下标 iv 和被乘数下标 iu 的初始值可按照如下公式设置[6]:

$$iv \leftarrow MIN(ir, vn-1) \tag{2}$$

$$iu = ir - iv \tag{3}$$

1.2 Comba 算法多核并行策略及实现

Comba 算法的核心操作为乘加运算与进位运算, 进位运算有较强的顺序性, 而乘加操作计算每个数位则具有较强的独立性. 为充分挖掘 Comba 算法的并行性, 可通过解决乘加运算与进位运算的数据相关性, 将乘加操作与进位操作分开执行, 并在乘加操作阶段采用 OpenMP 技术进行并行, 而进位操作阶段仍串行执行, 从而实现 Comba 算法的并行化. 并行化 Comba 算法如算法 1 所述.

算法 1　并行 Comba 算法

输入: 操作数 u、v, 及其位数 un、vn
输出: 乘积 r
1.为乘积 r 申请长度为 rn 的内存空间, 由指针 rp 指向, 其中 $rn = un + vn$
2.根据公式(1)及下述并行策略(1)、(2)循环求取中间结果 $(r_0, r_1, \ldots, r_{un+vn-1})$, 对应存储在临时数组中, 采用 OpenMP 的 for 任务分担策略进行并行, 线程数根据当前计算平台可用计算资源决定
3.基于步骤 2 所得 $(r_0, r_1, \ldots, r_{un+vn-1})$, 顺序对 $r_i(0 \le i < un+vn)$ 执行进位操作, 将最终结果写入 $rp[rn]$ 数组
4.计算乘积 r 的数位长度, 如果 $rp_{un+vn-1} != 0$, 则其长度为 $un+vn$; 否则, 其长度为 $un+vn-1$

其中具体并行策略如下所述:

1) 该算法的核心操作为 64 位无符号长整型数的乘加运算, 两个 64 位无符号整数相乘时, 结果为 127 或者 128 位, 而乘积 r 中间结果的每一个数位是一对或多对 64 位无符号长整型数相乘后相加的结果, 最终结果会大于或等于 127 位, 所以中间结果的数位需采用 3 个 64 位无符号长整型临时变量缓存. 若采用高级语言实现, 每进行一次乘加操作, 便需要进行两次进位判断, 因此采用内嵌汇编语言的方法实现此核心操作以消除每次求中间结果数位时的多次进位判断操作, 串

© 中国科学院软件研究所 http://www.c-s-a.org.cn

行进位操作也采用汇编语言实现.

2) 为实现乘加操作阶段的并行化, 需将乘加操作与进位操作分成两个独立的阶段执行, 即首先在乘加阶段求取并缓存中间结果, 此过程中不实施进位, 待中间结果求取完毕后统一实施进位操作. 为解决乘加操作与进位操作阶段的数据相关性问题, 并基于策略(1)所述, 可申请三个64位无符号长整型类型的数组存储通过乘加操作求得的中间结果. 设数组分别由指针 *lpl*, *hpl*, *cl* 指向, 则 *lpl[i], hpl[i], cl[i]*缓存乘积的中间结果的第 *i* 个数位, 依次为该数位的低 64 位, 中间 64 位以及高 64 位.

3) 乘加操作阶段, 基于中间结果每个数位的计算独立性, 通过 OpenMP 多线程编程将其并行化, 而进位阶段仍串行执行. 具体为在乘加操作阶段, 通过 OpenMP 多线程编程采用 for 任务分担策略实现, 将中间结果的数位计算同时分配给多个线程并行求取, 线程数目可根据计算机资源设定, 任务分割粒度根据实验获取. 本文基于 8 核实验平台, 开启 8 线程, 采用动态调度策略, 通过实验将任务粒度设为$rn/64$时效果最佳. 相比于静态调度策略, 动态策略更有利于负载均衡, 图 1 为当 *rn=22* 时采用静态策略和动态策略分配计算任务时对比的一种情况.

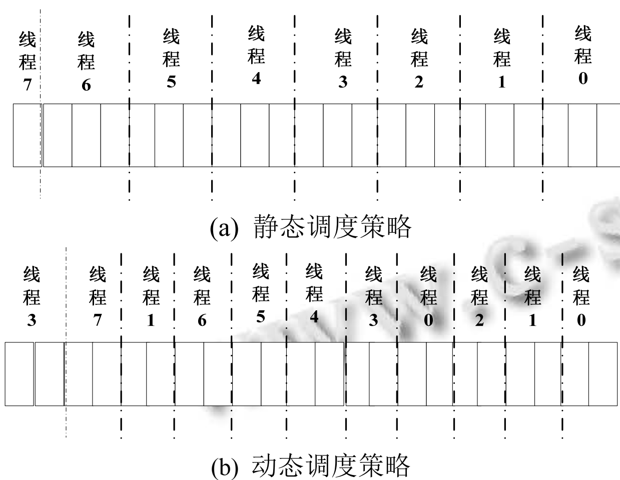

图 1 rn=22 时静态策略和动态策略任务调度对比

(4) 在程序编译时, 基于 SIMD 模型, 通过 Intel C++ Compiler 的自动向量化技术实现数据并行.

2 Karatsuba算法及其并行化

2.1 Karatsuba 算法描述

Karatsuba 算法基于分治思想, 假设基为 *R*, 乘数 *u*、被乘数 *v* 均为 *2n* 位, 拆分为两部分后分别为u_1、u_0和v_1、v_0, 位数均为 *n*, 则 *u*、*v* 可分别表示成如公式(4)、(5)所示:

$$u = u_1 * R^n + u_0 \tag{4}$$

$$v = v_1 * R^n + v_0 \tag{5}$$

由公式(4)和(5)以及代数技巧转化可得乘积 r 表示如下:

$$r = u_1 * v_1 * R^{2n} + [u_1 * v_1 + u_0 * v_0 - (u_1 - u_0)(v_1 - v_0)]R^n + u_0 * v_0 \tag{6}$$

基于公式(6), 可将 *2n* 位的大整数乘法转换成 3 个 *n* 位的大整数乘法以及若干次加法和减法运算, 可采用相同的策略求解规模逐渐减小的大整数乘法, 即递归实现, 该方法首次将大整数乘法的算法时间复杂度由$O(n^2)$降低为$O(n\log n)$.

2.2 Karatsuba 算法多核并行策略及实现

Karatsuba 算法基于分治思想, 将参与运算的乘数与被乘数的乘运算拆分成若干较小规模的部分积的计算, 每个部分积的计算可采用相同的策略递归求取, 而本文的并行策略便是基于部分积的计算独立性. 同时, 由于 Karatsuba 算法在实现时采用递归的形式, 而 CPU 资源有限, 如果开启的线程数较多, 处理器核进行线程调度时引起额外开销将是可观的, 因此并行化时需要考虑并行化深度. 本文并行化方案基于 8 核平台进行设计, 并行化时对原有Karatsuba算法进行改进, 并行深度为 1.

改进后的 Karatsuba 算法基于下述公式(9), 将初始乘数与被乘数的运算在递归算法第一层拆分为 8 个部分积的计算, 而在递归算法除第一层外的其它层, 即计算部分积时, 仍然调用原始串行 Karatsuba 算法.

设乘数 *u*、被乘数 *v* 的位数均为 *w*, 基仍设为 *R*, 将乘数 *u* 拆分成u_0、u_1、u_2 3 部分, 将被乘数 *v* 拆分成v_0、v_1、v_2 3 部分, u_0、u_1、v_0、v_1位数均为 *n*, u_2、v_2位数均为 *s*, 其中$n = w/3$, $s = w - 2*(w/3)$, 即 *u*、*v* 可分别表示成如公式(7)、(8)所示:

$$u = u_2 * R^{2n} + u_1 * R^n + u_0 \tag{7}$$

$$v = v_2 * R^{2n} + v_1 * R^n + v_0 \tag{8}$$

在拆分 *u*、*v* 时, 可借助大整数用数组存储数值的机制, 通过指针指向数据的高位及低位部分实现数据拆分, 从而减少重新申请数组而造成的空间浪费, 同时更加灵活高效.

基于公式(7)、(8)以及相应的代数转换技巧, 乘积

© 中国科学院软件研究所 http://www.c-s-a.org.cn

r 进而可转换为 8 个部分积及若干加减运算, 即如公式(9)所示.

$$r = r_4 * R^{4n} + r_3 * R^{3n} + r_2 * R^{2n} + r_1 * R^n + r_0 \quad (9)$$

其中: $r_0 = u_0 * v_0$, $r_1 = (u_1 + u_0) * (v_1 + v_0) - u_1 * v_1 - u_0 * v_0$, $r_2 = u_1 * v_1 + u_0 * v_2 + u_2 * v_0$, $r_3 = u_1 * v_2 + u_2 * v_1$, $r_4 = u_2 * v_2$.

本文 Karatsuba 算法基于改进后的公式(9)中 8 个部分积计算独立性, 通过采取适当的存储策略, 解决其数据相关性, 实现其并行化. 并行化 Karatsuba 算法如算法 2 所述.

算法 2 并行 Karatsuba 算法

输入: 被乘数 u、乘数 v,其位数均为 w

输出: u 与 v 的乘积 r

1.申请长度为 $2*n+2$ 的 64 位无符号长整型临时数组 $lowab[2*n+2]$;

2.通过大整数加运算将乘数 u 低 n 位与中间 n 位相加得到 $u_0 + u_1$, 并存储在步骤 1 申请的临时数组 $lowab$ 的低 $n+1$ 位; 将被乘数 v 的低 n 位与中间 n 位相加得到 $v_1 + v_0$, 临时数组的 $lowab$ 的高 $n+1$ 位;

3.根据下述并行方案中(1)、(2)所述并行计算并存储部分积 $u_0 * v_0$、$u_1 * v_1$、$u_2 * v_2$、$u_2 * v_0$、$(u_1 + u_0) * (v_1 + v_0)$、$u_0 * v_2$、$u_1 * v_2$ 以及 $u_2 * v_1$, 且数据 u_0、u_1、u_2 分别为乘数 u 的低 n 位, 中 n 位, 高 s 位, v_0、v_1、v_2 分别为被乘数 v 的低 n 位, 中 n 位, 高 s 位, 可通过指针指向获取, $u_0 + u_1$ 以及 $v_1 + v_0$ 由步骤 1、2 计算所得存储在临时数组 $lowab$ 中;

具体方案如下所述:

1) 申请长度为 $3*w+s+2$ 的 64 位无符号长整型临时数组 tmp. 根据公式(9), $u_0 * v_0$、$u_1 * v_1$、$u_2 * v_2$、$u_2 * v_0$、$(u_1 + u_0) * (v_1 + v_0)$、$u_0 * v_2$、$u_1 * v_2$ 以及 $u_2 * v_1$ 等 8 个部分积的计算并行执行, 而在求此 8 个部分积时, 仍然调用原始串行递归Karatsuba算法. 为解决其数据相关性问题, 采用如下策略进行存储, 其中 $u_0 * v_0$、$u_1 * v_1$、$u_2 * v_2$ 的计算结果分别存储在 rp 指向数组的低 $2n$, 中 $2n$ 位, 以及高 $2s$ 位, $u_2 * v_0$、$(u_1 + u_0) * (v_1 + v_0)$、$u_0 * v_2$、$u_1 * v_2$ 以及 $u_2 * v_1$ 的计算结果存储在申请的临时数组 tmp 中, 存储策略如图 2 所示, 其中 rp 为存储最终乘积 r 的数组;

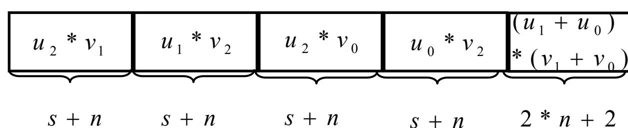

图 2 临时数组 tmp 存储部分积示意图

2) 由策略(1)所述在递归算法第一层计算 8 个部分积时, 采用 OpenMP 并行编程技术中 section 并行策略进行并行化, 开启 8 个计算线程并行计算, 每个线程执行一个部分积的计算过程, 且计算结果根据策略(1)所述方案存储;

3) 由策略(1)和策略(2)所述得到的 8 个部分积的计算结果, 分别存储在数组 rp 以及临时数组 tmp 中, 根据公式(9)将存储在数组 rp 以及临时数组 tmp 中的部分积进行加减归并运算, 此过程串行执行, 并将最终结果存储在数组 rp 中;

采用此策略可使数据计算以及存储均无重叠情况, 在并行计算时无数据相关性问题, 可完全并行执行, 归并时基于公式(9)进行串行归并. 并行策略如图 3 所示.

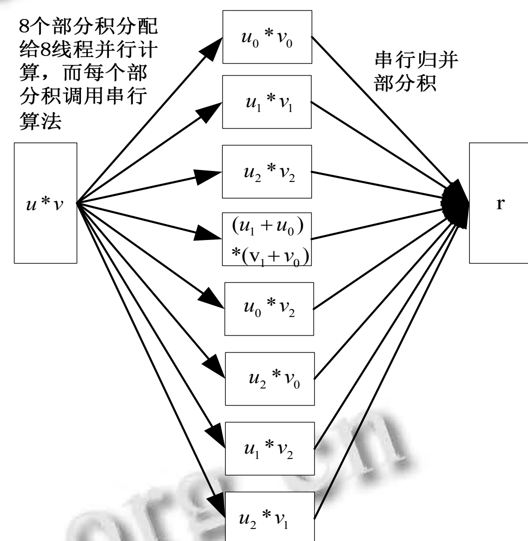

图 3 Karatsuba 并行策略示意图

3 性能测试与分析

本文实验测试平台为 8 核 Intel Xeon 55 系列处理器, 主频为 2.67GHz, 以 Intel C++ Compiler 11.1 为编译器, C 语言为开发语言, 基为 2^{64}, 实现了串并行 Comba 算法以及 Karatsuba 并行算法, 其中正确性验证采用与被广泛应用的大整数开源数学库 GMP5.1.3 版本的运行结果进行对比的方式[7], 串行 Comba 算法主要用于并行算法性能对比测试, 并行 Karatsuba 算法的性能对比测试程序直接调用 GMP 库中的串行 Karatsuba 函数进行对比. 测试数据中, 并行 Comba 及 Karatsuba 算法均开启 8 线程并行, 被乘数与乘数规模为以 64 位无符号长整型数据类型为单位, 程序运行时间单位为微秒.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

3.1 并行 Comba 算法测试结果

串行 Comba 与并行 Comba 部分测试数据运行时间对比如图 4 所示, 其中 *seq.*、*p.l* 分别代表串、并行程序. 由于 Comba 算法是常在数据规模较小时调用的大整数乘法, 因此本文图例展示了数据规模较小时测试的加速结果. 并行 Comba 算法与串行 Comba 算法的加速比平均可达到 5.63, 从测试数据中可以看到在数据规模较小时便可以达到相当的加速比, 随着数据规模的增大, 加速比并未明显提升. 原因在于虽然乘加运算具有完全的并行度, 但是由于乘加操作计算完毕后, 进位操作需要进行串行归并, 因此随着数据规模的增大, 其加速比未能大幅增加. 相比于 Vladislav Kovtun 和 Andrew Okhrimenko 所采取的策略, 本文在负载均衡以及并行度上均有所改进.

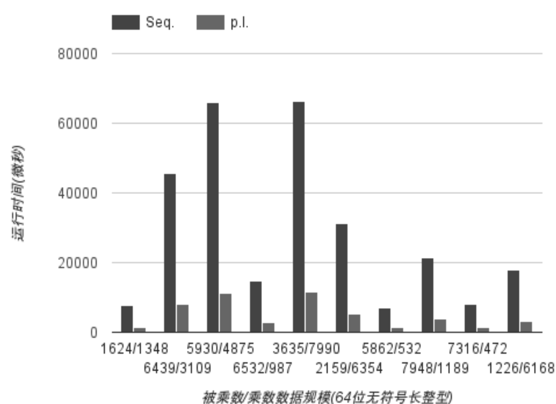

图 4　串行 Comba 与并行 Comba 运行时间对比图

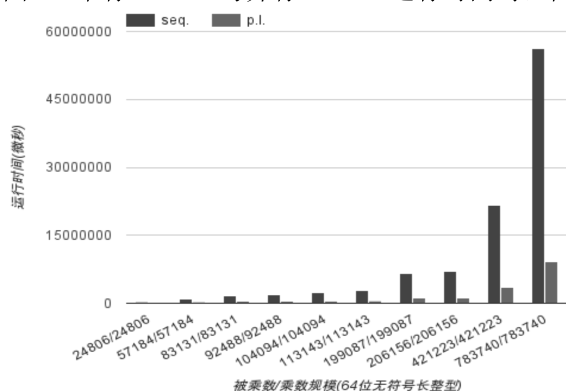

图 5　串行与并行 Karatusba 算法运行时间对比图

3.2 并行 Karatsuba 算法测试结果

串行与并行 Karatusba 算法运行时间对比如图 5 所示, 其中 *seq.*、*p.l* 分别代表串、并行程序. 并行 Karatsuba 算法与串行 Karatsuba 算法的加速比平均可达到 5.12, 最高时加速比为 6.14, 且从图 5 中可以看出随着数据规模的增大, Karatsuba 算法的加速比逐渐增加. 该并行算法中, 8 个部分积可完全并行计算, 相比于 Comba 算法, Karatsuba 算法串行计算部分所占比例较小, 尤其随着数据规模的增大, 算法并行计算的部分占总运行时间的比例增大, 加速比逐渐提升.

4　结语

本文基于 64 位 8 核对称多核处理机, 进行了基为 2^{64} 的大整数乘法 Comba 算法和 Karatsuba 算法的多核并行化研究, 采用了多种优化策略, 如: 在代码级别对部分核心代码直接采用汇编语言实现, 采用数据向量化进行并行, 而在线程级别通过降低数据之间的相关性进而采用 OpenMP 多线程编程技术进行并行化, 通过这三种优化技术, 两种快速算法均取得了较为理想的性能提升.

参考文献

1 陈智敏.RSA 公钥体制中快速大整数乘法的实现.广州大学学报,2002,1(3):44–45.

2 Kovtun V, Okhrimenko A. Approaches for the parallelization of software implementation of integer multiplication. IACR Cryptology ePrint Archive, 2012, 2012: 482.

3 Kovtun V, Okhrimenko A. Approaches for the performance increasing of software implementation of integer multiplication in prime fields. IACR Cryptology ePrint Archive, 2012: 170.

4 Jebelean T. Using the parallel Karatsuba algorithm for long integer multiplication and division. Euro-Par’97 Parallel Processing. Springer Berlin Heidelberg, 1997: 1169–1172.

5 圣丹斯.BigNumMath:加密多精度算法的理论与实现.北京:水利水电出版社,2008.

6 桑波.大整数乘法算法的研究与实现[硕士学位论文].广州:华南理工大学,2012.

7 Granlund T and the GMP development team. GNU MP--The GNU Multiple Precision Arithmetic Library, Edition 6.0.0., 25 March 2014.

8 Buchberger B, Jebelean T. Systolic multiprecision arithmetic. Impact Tempus Jep and Hungarian Transputer Users Group Workshop Parallel Processing in Education. 1993.

© 中国科学院软件研究所　http://www.c-s-a.org.cn