Capítulo 10

Implementación y resultados experimentales

10.1. Introducción

Con el objeto de hacer pruebas de desempeño del algoritmo paralelo que presentamos en el capítulo previo, así como comparaciones con los algoritmos de multiplicación escalar presentados en el capítulo 8, se llevó a cabo la implementación de la aritmética de puntos sobre una curva elíptica. La implementación se realizó sobre una multicomputadora de componentes comunes, utilizando las librerias de interfaz de paso de mensajes MPI. En este capítulo describiremos las características de la multicomputadora utilizada, los detalles de implementación de las operaciones de la curva elíptica y del campo subyacente, el tipo de pruebas realizadas y los resultados de las pruebas.

10.2. Implementación del campo subyacente

Como campo subyacente se eligió $GF(2^m)$ eligiendo m de tal forma que exista una curva de Koblitz de orden casi primo sobre $GF(2^m)$. Dado que el orden de $E(GF(2^l))$ divide al orden de $E(GF(2^m))$ si l divide a m, esto impone el requerimiento de que m sea primo. Tomamos por tanto $m = 163$, 233, 283, 409 o 571.

Las implementaciones más simples y rápidas de la aritmética de campo finito se han obtenido utilizando representaciones en base polinomial con un trinomio o pentanomio como polinomio irreducible [106]. Hemos por lo tanto utilizado representaciones polinomiales de ese tipo.

Sea $f(x) = x^m + r(m)$ un polinomio binario irreducible de orden m. Los elementos de $GF(2^m)$ son representados por los polinomios binarios de grado a lo más $m-1$ con suma y multiplicación módulo $f(x)$.

Un elemento del campo $a(x) = a_{m-1}x^{m-1} + \cdots + a_2x^2 + a_1x + a_0$ es asociado con el vector binario $a = (a_0a_1a_2 \ldots a_{m-2}a_{m-1})$. Obsérvese que los bits más significativos están más a la derecha de manera opuesta a como se especifican en el estándar del NIST [73]. Esto se hace así por requerimientos de la libreria NTL, que mencionaremos posteriormente.

Sea $t = \lceil m/8 \rceil$ y $s = 8t - m$, entonces en la implementación almacenamos a como el arreglo de t bytes $A = (A[0], A[1], \ldots, A[t-1], A[t])$ donde los s bits más a la derecha de $A[t]$ son puestos en 0. Para lectura y escritura, A es formateado como una secuencia de dígitos hexadecimales que representan $A[i]$ para $i = 0, 1, \ldots, m-1$.

Para implementar la aritmética del campo finito hemos utilizado la clase *GF2E* de la libreria NTL. Para mayor información sobre NTL y su implementación ver [74].

10.3. Implementación de curvas elípticas

Las curvas elípticas elegidas para las pruebas fueron algunas de las recomendadas por el NIST [73]. El estándar del NIST incluye diez campos finitos recomendados: cinco campos primos y cinco campos binarios. Los campos binarios son $GF(2^m)$ donde m = 163, 233, 283, 409 o 571. Para cada uno de los campos primos, se recomienda una curva elíptica elegida aleatoriamente, mientras que para cada campo binario son recomendadas una curva binaria aleatoria y una curva de Koblitz.

Los elementos de $GF(2^m)$ son representados usando una representación en base polinomial con polinomio de reducción $f(x)$, tal como se detalló en la sección anterior. Una curva elíptica E definida sobre $GF(2^m)$ es especificada por los coeficientes $a, b \in GF(2^m)$ de su ecuación $y^2 + xy = x^3 + ax^2 + b$. El número de puntos sobre E es nh donde n es primo y h es denominado el cofactor. Una curva aleatoria sobre $GF(2^m)$ es denotada como B-m, mientras que una curva de Koblitz definida sobre $GF(2^m)$ es denotada como K-m. Algunas de estas curvas se muestran en la tabla 10.3.

Para representar los puntos sobre una curva elíptica se utilizaron coordenadas afines [92]. Se realizó una implementación orientada a objetos de la aritmética de puntos sobre curvas elípticas, cuyo código se muestra en el apéndice A. Los puntos sobre una curva se representan como objetos de la clase *ec_point*, la cual se encuentra parametrizada por el grado de exten-

B-163	$f(x) = x^{163} + x^7 + x^6 + x^3 + 1$ $a =$ 0x1 $b =$ 0xdf5023a44787f21501be1841ac359c8b709106a02 $h = 2$ $n = 5846006549323611672814742442876390689256843201587$
B-233	$f(x) = x^{233} + x^{74} + 1$ $a =$ 0x1 $b =$ 0xda09f8d7f511ef1824ec9e02b333b31285bb3290c8f7c233c 6ede746660 $h = 2$ $n = 6901746346790563787434755862277025555839812737345 013555379383634485463$
K-163	$f(x) = x^{163} + x^7 + x^6 + x^3 + 1$ $a =$ 0x1 $b =$ 0x1 $h = 2$ $n = 5846006549323611672814741753598448348329118574063$

Cuadro 10.1: Curvas elípticas sobre campos binarios

sión m del campo binario subyacente, el polinomio irreducible $\phi(x)$, y los coeficientes a y b de la ecuación de la curva, permitiendo así que el campo subyacente y la curva elíptica a utilizar puedan especificarse a tiempo de ejecución.

Entre los métodos de la clase se incluyen aquellos necesarios para calcular el negativo de un punto, la suma de puntos, la duplicación y multiplicación de puntos, para probar si dos puntos son iguales, para verificar que un punto pertenezca a la curva, etc.

10.4. Desempeño de las operaciones de puntos

Se midieron los tiempos de ejecución de las operaciones de puntos de curvas elípticas implementadas en la libreria *ECC*. Para cada valor de m, tanto en la curva aleatoria como en la de Koblitz[1], se midió el tiempo que se requiera para realizar una suma de puntos, una duplicación y una multiplicación. La computadora utilizada para realizar las pruebas fue una computadora personal con procesador Pentium III a 600 MHz. Si bien al momento

[1]Ver parámetros de las curvas utilizadas en el apéndice E.

de realizar las pruebas ya estaba disponible el procesador Pentium 4, utilizamos el anterior debido a que el *cluster* en que se realizaron las pruebas del algoritmo paralelo está construido con procesadores similares.

Los tiempos de ejecución (en segundos) obtenidos se resumen en la siguiente tabla:

curva	**suma**	**dupl**	**mult**
K-163	15×10^{-5}	7×10^{-5}	0.023
K-233	21×10^{-5}	11×10^{-5}	0.047
K-283	28×10^{-5}	14×10^{-5}	0.078
K-409	45×10^{-5}	22×10^{-5}	0.18
K-571	76×10^{-5}	38×10^{-5}	0.424
B-163	15×10^{-5}	8×10^{-5}	0.023
B-233	21×10^{-5}	11×10^{-5}	0.047
B-283	29×10^{-5}	14×10^{-5}	0.077
B-409	45×10^{-5}	23×10^{-5}	0.179
B-571	76×10^{-5}	37×10^{-5}	0.423

Como puede observarse, el tiempo requerido para realizar una suma es el doble del requerido para realizar una duplicación. También vemos que los tiempos de ejecución son los mismos, utilizando el algoritmo binario, en las curvas aleatorias y las de Koblitz.

Los tiempos obtenidos son mejores a los reportados en [88] (0.092 segundos para una multiplicación de 155 bits), [106] (0.072 seg. para 176 bits), [40] (0.068 seg. para 176 bits) y cercanos a los reportados en [60] (0.0135 seg. para 163 bits) aún cuando fueron utilizadas UltraSPARC [60] y Dec ALPHA 3000 [106] como plataformas.

10.5. Implementación del algoritmo paralelo

El algoritmo binario de orden p fue programado en C++ utilizando la libreria *ECC* de aritmética de puntos, así como la libreria de paso de mensajes MPI. El código de la implementación se incluye en el apéndice D.

La máquina utilizada para realizar la implementación del algoritmo paralelo fue una multicomputadora de componentes comunes construida exprofeso. Fue ensamblada en su totalidad con componentes de uso común en las computadoras personales, siguiendo el diseño utilizado en un *cluster Beowulf* [11].

La multicomputadora está compuesta de 3 nodos: Un nodo maestro y dos nodos esclavos. Cada nodo tiene un Motherboard de Procesador Dual

Pentium III a 600 MHz, por lo tanto la multicomputadora tiene en total 6 procesadores.

Las características del nodo maestro son las siguientes:

- Procesador Dual Pentium III 600 MHz
- 512 MB de memoria RAM
- Disco duro de 20 Gb.
- Tarjeta de red Realtek Fast Ethernet

Mientras que las características de los nodos esclavos son las siguientes:

- Procesador Dual Pentium III 600 MHz
- 128 Mb de memoria RAM
- Sin disco duro
- Tarjeta de red Intel PRO100 con PXE

Los nodos fueron interconectados mediante un Hub 3Com Office Conect de 100 Mb/s.

El sistema operativo utilizado fue el RedHat Linux 6.2 con kernel Linux SMP 2.2.16. Para la programación se utilizo el compilador GNU de lenguaje C++ y las librerias de paso de mensajes LAM/MPI.

10.6. Desempeño del algoritmo paralelo

10.6.1. Curvas binarias aleatorias

En primer lugar se midieron tiempos de ejecución para la operación de multiplicación escalar sobre las curvas B-m. En el apéndice E se detallan los parámetros de las curvas utilizadas.

Los tiempos de ejecución (en segundos) del algoritmo paralelo se muestran en la tabla siguiente:

m	p=2	p=4	p=8	p=16
163	0.0185	0.0156	0.0142	0.01359
233	0.0369	0.0310	0.0281	0.02683
283	0.0597	0.0500	0.0454	0.04322
409	0.1385	0.1159	0.1049	0.09958
571	0.3262	0.2727	0.2464	0.23358

Hay que señalar que el parámetro p indicado, no es realmente el número de procesadores utilizados, el cual está fijo en 6, sino el número de *procesos* generados en paralelo en la ejecución en el entorno MPI.

A partir de los tiempos medidos, las *tasas de aceleramiento* obtenidas son las que se muestran a continuación.

m	S_2^*	S_4^*	S_8^*	S_{16}^*
163	1.2441	1.4761	1.6194	1.6925
233	1.2735	1.5161	1.6694	1.7515
283	1.3063	1.5574	1.7176	1.8049
409	1.2998	1.5526	1.7163	1.8077
571	1.2997	1.5546	1.7209	1.8152

De la tabla anterior, podemos observar que, para este tipo de curvas, no se tiene un aceleramiento lineal, sino que el aceleramiento decae rápidamente conforme aumenta el número de procesos. Aún asi, estos resultados son comparables al aceleramiento de 1.25 que podriamos esperar utilizando métodos de ventanas [53][51].

Los resultados de *eficiencia por procesador* se muestran en la siguiente tabla.

m	E_2^*	E_4^*	E_8^*	E_{16}^*
163	0.6220	0.3690	0.2024	0.1058
233	0.6367	0.3790	0.2087	0.1095
283	0.6532	0.3893	0.2147	0.1128
409	0.6499	0.3882	0.2145	0.1130
571	0.6498	0.3886	0.2151	0.1135

Debido a que estos indicadores no son cercanos a 1, valor que indica optimalidad, podemos concluir que el algoritmo binario de orden p no es muy eficiente al utilizar curvas binarias aleatorias. Este resultado es de esperarse debido a que, en primer lugar, como se dedujo en el capítulo anterior, en el caso general el número de duplicaciones necesarias permanece igual al distribuir el cálculo de múltiplos entre los procesadores y en segundo lugar, la comunicación entre procesos introduce un costo extra en el tiempo de ejecución.

10.6.2. Curvas de Koblitz

En esta sección presentaremos los resultados obtenidos utilizando el algoritmo τ-ario de orden p sobre curvas de Koblitz. Resultados previos fueron presentados en [36].

Para obtener los resultados siguientes, se generaron secuencias binarias aleatorias que se tomaron como representaciones de enteros en base τ y se evaluaron los correspondientes múltiplos de puntos. Los parámetros de las curvas utilizadas se detallan en el apéndice E.

Los tiempos medidos para las curvas K-m son los que se enumeran a continuación:

m	p=2	p=4	p=8	p=16
163	0.0063	0.0034	0.0020	0.0014
233	0.0124	0.0065	0.0037	0.0024
283	0.0201	0.0105	0.0058	0.0036
409	0.0465	0.0239	0.0129	0.0076
571	0.1093	0.0558	0.0294	0.0166

Los anteriores resultados nos proporcionan las siguientes *tasas de aceleración*:

m	S_2^*	S_4^*	S_8^*	S_{16}^*
163	1.8220	3.400	5.7681	8.3647
233	2.0599	3.921	6.9493	10.8186
283	1.9721	3.786	6.8399	11.0170
409	1.9366	3.764	7.0006	11.9154
571	1.9861	3.891	7.3796	13.0701

Por lo tanto obtenemos los siguientes resultados de *eficiencia por procesador*:

m	E_2^*	E_4^*	E_8^*	E_{16}^*
163	0.9110	0.8499	0.7210	0.5228
233	1.0299	0.9803	0.8687	0.6762
283	0.9861	0.9465	0.8550	0.6886
409	0.9683	0.9410	0.8751	0.7447
571	0.9930	0.9727	0.9225	0.8169

Podemos observar que los valores resultantes son cercanos a 1, por lo tanto son cercanos a lo óptimo. Tomando en consideración que en la implementación de cualquier algoritmo paralelo hay un costo de tiempo asociado a la comunicación entre procesos, podemos afirmar que en el caso de las curvas de Koblitz, nuestro algoritmo es óptimo.

10.7. Conclusiones

Se ha presentado en este trabajo una generalización del algoritmo binario que, utilizando procesamiento paralelo, puede acelerar el cálculo de la multiplicación escalar sobre curvas elípticas.

Se ha llevado a cabo la implementación del algoritmo sobre una multicomputadora de tipo Beowulf. Los tiempos de ejecución medidos han sido más cortos a los reportados en la literatura para otras implementaciones en software.

Se ha mostrado como este algoritmo es particularmente eficiente en el caso de las curvas de Koblitz, en las cuales se logra un *speed up* lineal.