Utilizando CORBA para distribuir aplicaciones intensivas en cálculo.

Jorge Enrique Pérez Martínez y Esperanza Marcos Martínez

Universidad Politécnica de Madrid
Departamento de Informática Aplicada
Campus Sur de la U.P.M., 28031 Spain
email: jeperez@eui.upm.es

Universidad Rey Juan Carlos
E.S. de Ciencias Experimentales y Tecnología
Campus de Móstoles, 28933 Spain
email: cuca@escet.urjc.es

Abstract.

Actualmente se están utilizando supercomputadores para resolver problemas intensivos en cálculo tales como las previsiones meteorológicas, el aprendizaje de redes neuronales artificiales o incluso aplicaciones lúdicas como el ajedrez. El uso de esta tecnología queda limitado a las grandes corporaciones con medios financieros suficientes como para abordar la adquisición de tales ordenadores. La distribución del cálculo entre varios ordenadores es una solución más barata para aquellos casos en los que la aplicación es susceptible de ser distribuida. De hecho, es la única solución para muchas empresas que no pueden adquirir supercomputadores y que sin embargo soportan aplicaciones que precisan de mucha potencia de cálculo (por ejemplo, el análisis de imágenes). En este artículo mostraremos cómo utilizando un middelware, CORBA (Common Object Request Broker Architecture), y una implementación concreta de este, DST (Distributed Smalltalk), es posible distribuir una aplicación entre varios ordenadores de una manera elegante y escalable y cómo el trabajo cooperativo de varios ordenadores disminuye significativamente el tiempo total de cómputo. Creemos que esta forma de distribución puede solucionar muchos de los problemas de tiempos de ejecución con los que se enfrentan actualmente las empresas de desarrollo software.

Palabras clave: objetos distribuidos, CORBAservices, ORB, DST, mensajes asíncronos, UML.

1. Introducción.

Hoy en día estamos construyendo aplicaciones software que antaño, y por falta de potencia de cálculo, eran inabordables. Aún así, la complejidad algorítmica de determinados problemas impide que su solución se obtenga en el intervalo de tiempo adecuado (por ejemplo, el cálculo de la siguiente jugada en una partida de ajedrez). La construcción de supercomputadores especializados ha permitido resolver parcialmente estos problemas. Sin embargo, el elevado precio de los mismos constituye una barrera insalvable para muchas organizaciones que además no necesitan tanta potencia de cálculo pero sí bastante más que la proporcionada por un procesador convencional. Si tenemos presente que los modelos menos potentes de estos supercomputadores cuestan decenas de millones

de pesetas, se puede concluir que es una solución más barata la compra e interconexión de decenas de ordenadores convencionales (tipo PC).

En este artículo mostramos, mediante un ejemplo -cálculo de fractales-, cómo abordar problemas intensivos en cálculo mediante el uso de varios ordenadores convencionales. La forma de distribuir la aplicación debe ser transparente y escalable y debe ofrecer ganancia de rendimiento en términos de tiempo total de cálculo. El problema del cálculo de fractales es susceptible de ser distribuido: cada ordenador ejecutará el cálculo sobre un subconjunto de puntos del plano complejo. La distribución es escalable: cuantos más puntos a calcular más ordenadores podemos introducir sin variar en absoluto el diseño software de la solución. Para implementar la distribución haremos uso de una implementación de CORBA: DST. DST [1] soporta una implementación de los estándares del OMG (Object Management Group) tanto del ORB (Object Request Broker) (CORBA 2.0 [4]) como del CORBAservices [3]. Justamente son los servicios de nombrado y de ciclo de vida los que nos proporcionarán la transparencia en la localización.

En la sección 2 presentamos el caso de estudio. La sección 3 propone una solución al problema. En la sección 4 ofrecemos el conjunto de medidas obtenidas al ejecutar la aplicación de forma centralizada y de forma distribuida. Por último, en la sección 5 presentamos las conclusiones.

2. Caso de estudio: cálculo y representación de fractales.

Los fractales son representaciones gráficas de funciones matemáticas. Lo que se abordará aquí es el cálculo y la representación de un fractal conocido como conjunto de Mandelbrot [2]. Dicho conjunto está compuesto por los números complejos c=a+ib, para los que la relación de recurrencia:

$$Z_{n+1} = Z_n^2 + c \quad \text{para} \quad n = 0, 1, 2, ... \qquad (1)$$

converge a un número complejo finito, siendo $Z_n = 0$ la condición inicial.

Se demuestra que sí existe un "n" para el cual $|Z_n| > 2$, la serie diverge y "c" no pertenece al conjunto de Mandelbrot. Como el "n", de existir, podría ser muy grande, la aproximación que se hace es llegar como máximo hasta el elemento Z_m. Si se alcanza Z_m sin que ningún $|Z_i| > 2$, se asume que "c" pertenece al conjunto de Mandelbrot.

El desarrollo de la serie de recurrencia es el siguiente:

$$X_{n+1} = X_n^2 - Y_n^2 + a; \quad Y_{n+1} = 2X_nY_n + b \qquad (2)$$

El fractal lo representaremos sobre una ventana. Si asociamos cada pixel (de la ventana en la que se realizará el dibujo) con un punto del plano complejo, podemos evaluar la serie de recurrencia para todos estos puntos y dibujar de un color (negro) los pixels que se correspondan con puntos del plano complejo que pertenezcan al conjunto de Mandelbrot y de otro color (blanco) los demás. Como puede suponerse, esta es una aplicación con un perfil "cpu-bound". Cuanto mayor sea el número de pixels de la ventana de dibujo mayor será el número de puntos del plano complejo a estudiar. Cuanto mayor sea el número de iteraciones por cada punto, mayor será el tiempo de cálculo. Para disminuir estos tiempos se va a realizar una distribución de la aplicación. Un objeto soportará la interacción del usuario con la aplicación (introducción de los datos del fractal, selección de la zona de "zoom", selección de la escala, etc., así como la visualización del fractal resultante del cálculo) mientras que el resto de objetos realizarán parte del cálculo del fractal y comunicarán sus resultados al objeto que interacciona con el usuario.

3. Solución propuesta al problema.

Utilizaremos UML [6] para representar los diferentes aspectos de la aplicación. En la figura 1 se ilustran las clases significativas que intervienen en la solución.

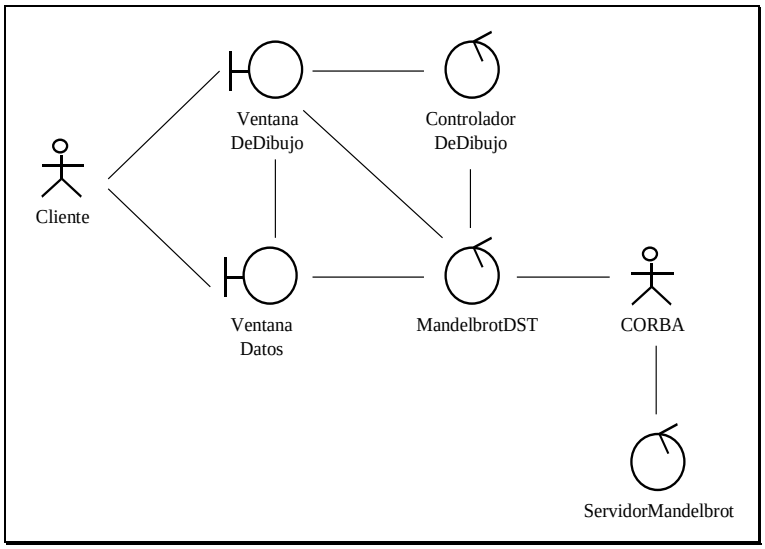

Figura 1. Clases en la solución al problema de los fractales.

Las clases *VentanaDeDibujo*, *ControladorDeDibujo* y *VentanaDatos* soportan la interfaz de la aplicación. Las clases *MandelbrotDST* y *SevidorMandelbrot* constituyen los objetos que interaccionarán para realizar los cálculos del fractal. El actor "Cliente" representa al usuario que quiere calcular y representar el fractal; el actor "CORBA" representa a un subsistema externo que ofrece los servicios distribuidos de un middelware. Nos centraremos en las clases *MandelbrotDST* y *ServidorMandelbrot* y obviaremos el resto del proceso de desarrollo así como los aspectos de estas clases que no intervienen en la distribución de la aplicación.

La clase *MandelbrotDST* será la responsable de:

- Obtener los datos introducidos por el usuario relativos al fractal a calcular.
- Distribuir el cálculo del fractal sobre objetos servidores (instancias de la clase *ServidorMandelbrot*). A cada objeto servidor se le encomienda el cálculo de n filas de la ventana dónde se dibujará el fractal.
- Visualizar en una instancia de *VentanaDeDibujo* los resultados devueltos por los objetos servidores.

Por su parte, la clase *ServidorMandelbrot* soportará sólo una responsabilidad:

- Cálculo de n filas de un fractal.

Para implementar este modelo haremos que la única instancia de la clase *MandelbrotDST* (inicialmente cliente) invoque asíncronamente el método de cálculo sobre n instancias de la clase *ServidorMandelbrot* (inicialmente servidor). Mientras realiza tales invocaciones, algún servidor (una de las n instancias) habrá concluido sus cálculos y entonces invocará de manera asíncrona un método del cliente (instancia de la clase *MandelbrotDST*) que se encargará de visualizar los resultados en una ventana del cliente. Si el servidor no tiene nada más que hacer después de realizar los cálculos, la invocación del método del cliente puede ser síncrona. La invocación del método del cliente por parte del servidor es lo que conocemos por call-back y lo que soporta, por ejemplo, NewiORB [5].

A modo de ejemplo, crearemos dos instancias del servidor en dos ordenadores distintos (de nombre servidor1 y servidor2) mientras que el cliente se ejecutará en el ordenador 'cliente'. El diagrama de despliegue resultante se muestra en la figura 2. Para implementar esta solución en DST lo primero que debemos hacer es generar las interfaces IDL (Interface Definition Language) para las clases *MandelbrotDST* y *ServidorMandelbrot*.

El siguiente paso es establecer que el servidor1 (por ejemplo) contendrá el servicio de nombrado. Ahora hay que "levantar" un ORB en cada uno de los tres ordenadores. A continuación, en el servidor1 hay que dar de alta, en el servicio de nombrado, al FactoryFinder de ese servidor. En el servidor1 ejecutamos:

```
ns:= ORBObject namingService.
ns contextBind:'ffservidor1' asDSTName
        to:(ORBObject resolveInitialReferences:#FactoryFinder).
```

Lo mismo hay que hacer en el servidor2:

```
ns:= ORBObject namingService.
ns contextBind:'ffservidor2' asDSTName
        to:(ORBObject resolveInitialReferences:#FactoryFinder).
```

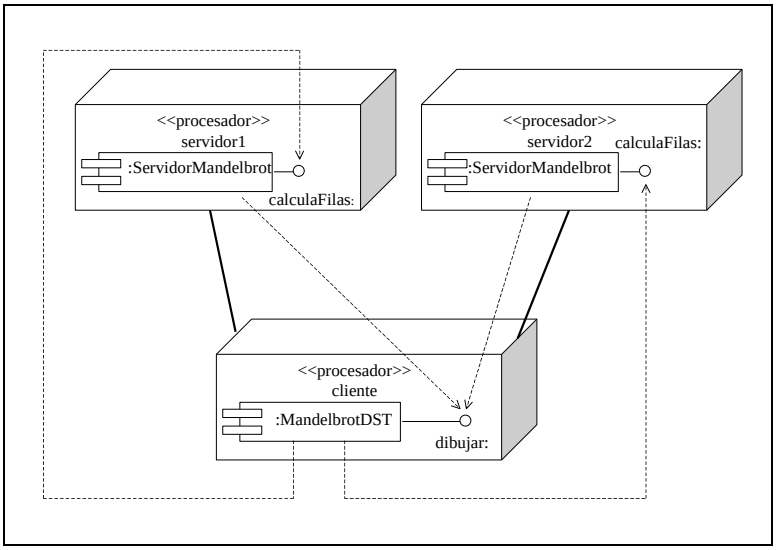

Figura 2. Diagrama de despliegue.

Ahora ya tenemos en el servicio de nombrado referencias a los objetos FactoryFinder's de los dos servidores. Desde el cliente podemos crear una instancia remota de la clase *ServidorMandelbrot* (por ejemplo en el servidor2) haciendo:

```
ns:= ORBObject namingService.
ff:= ns contextResolve:(DSTName onString:'ffservidor2').
clase:= (ff findFactoriesKey:('ServidorMandelbrot' asDSTName)) at:1.
servidor:= ff createObject:clase getInstanceACL.
```

A partir de ahora, al objeto remoto (servidor) se le pueden enviar peticiones de manera asíncrona. Vamos a utilizar las facilidades del DII (Dynamic Invocation Interface). Para ello construiremos primero, y de forma dinámica, los argumentos que van en el mensaje a enviar:

```
filas:= Interval from:0 to:100. "por ejemplo"
argumentos:= ORBNVList new initialize.
argumentos
    addNamedValue:(DSTNamedValue name:'filas' value:filas flags:0);
    addNamedValue:(DSTNamedValue name:'origenR' value:origenReal flags:0);
    addNamedValue:(DSTNamedValue name:'origenImag' value:origenImaginario flags:0);
    addNamedValue:(DSTNamedValue name:'rangoX' value:anchoReal flags:0);
    addNamedValue:(DSTNamedValue name:'rangoY' value:altoImaginario flags:0);
```

```
addNamedValue:(DSTNamedValue name:'ancho' value:ancho flags:0);
addNamedValue:(DSTNamedValue name:'alto' value:alto flags:0);
addNamedValue:(DSTNamedValue name:'bits' value:5 flags:0);
addNamedValue:(DSTNamedValue name:'Niteraciones' value:iteraciones flags:0);
addNamedValue:(DSTNamedValue name:'quien' value:self  flags:0);
addNamedValue:(DSTNamedValue name:'simbolo' value:#dibujar: flags:0).
```

Obsérvese que los dos últimos argumentos contienen la identidad del cliente (self) y el método del cliente (#dibujar:) a invocar por el servidor cuando finalice el cálculo. Ahora el cliente puede construir el mensaje con los argumentos anteriores:

```
selec:=:#calculaFilas:origenReal:origenImag:rangoX:rangoY:ancho:alto:bits:iteraciones:yo:
        operación:
mensaje:= servidor
                createRequestIn:#()
                selector:selec
                withArguments:argumentos
                result: (Array new:4)
                flags:0
```

y mandarlo de manera asíncrona:

```
(mensaje at:#request) send.
```

En el lado del cliente lo único que queda por hacer es esperar a que todas las peticiones enviadas hayan sido contestadas. Pero simultáneamente, también se están produciendo invocaciones desde los dos servidores a un método definido en el cliente (#dibujar:). Este método se limita a visualizar por la pantalla del cliente el resultado de los cálculos y a eliminar la petición a la que corresponde esta respuesta de la lista de peticiones pendientes.

En cuanto a los servidores, el único método que hacen público es el de cálculo y lo único reseñable es la forma de invocar el call-back al final del mismo:

```
quien perform:simbolo with:solucion.
self destroy.
```

donde quien representa al cliente, simbolo representa el método a invocar en el cliente y solucion representa los resultados del cálculo y que serán visualizados por el cliente. Comentar que perform es una llamada síncrona y que quien es un objeto remoto. Por último el servidor se autodestruye invocando una llamada del servicio del ciclo de vida (destroy).

5. Resultados de las ejecuciones

Se han efectuado mediciones sobre el tiempo de ejecución del cálculo de fractales en forma centralizada y en forma distribuida (con uno y varios servidores). Las condiciones de contorno de tales mediciones son las siguientes:

1) No se contabiliza el tiempo invertido en dibujar los resultados del cálculo en pantalla; sólo se contabilizan los tiempos de cálculo.
2) En cada Pentium Dual tan sólo está levantado un ORB. Sólo hay un proceso y no se pueden utilizar los dos procesadores
3) No se considera el tráfico de la red y la carga local de cada ordenador (en la versión distribuida). Para soslayar la carga impuesta por procesos del sistema operativo en un momento dado (procesos de XWindows, garbage collector, estado de la memoria, demonios, etc.) y/o procesos de usuario, hemos efectuado varias ejecuciones con los mismos datos y obtenido la media aritmética de las mismas.
4) No se controla cuando entra el garbage collector o cuando se planifican los demonios o cuándo se generan faltas de página.

5) Los perfiles hardware para las mediciones han sido los mostrados en la tabla 1 (la configuración-A corresponde a la versión centralizada y todas las demás a las versiones distribuidas). En todas las configuraciones se ha utilizado el sistema operativo Linux Red Hat 6.0.

Tabla-1	cliente	servidor1	servidor2	servidor3
configuración-A		X		
configuración-B	X	X		
configuración-C	X	X	X	
configuración-D	X	X	X	X

cliente:	Pentium	133Mhz,	32Mb RAM
servidor1:	Pentium II Dual	400Mhz,	256Mb RAM
servidor2:	Pentium	350Mhz,	512Mb RAM
servidor3:	Pentium II Dual	400Mhz,	128Mb RAM

Tabla 1. Configuraciones utilizadas para ejecutar el cálculo de fractales.

Se han realizado mediciones sobre 8 zonas distintas del plano complejo con diferentes tamaños de ventanas y diferentes números de iteraciones. En el apéndice-A aparecen los diferentes fractales calculados. En la tabla-2 se muestra el resultado en milisegundos para cada una de las 8 zonas y configuraciones. La figura 3 ilustra de manera gráfica estos mismos resultados.

Tabla-2	**Fig.1**	**Fig.2**	**Fig.3**	**Fig.4**	**Fig.5**	**Fig.6**	**Fig.7**	**Fig.8**
Conf-A	9751	32353	53114	18169	50596	45326	159351	133450
Conf-B	10044	34421	53757	19473	51304	45989	159730	135909
Conf-C	5668	16439	27479	9920	24733	24703	83930	66559
Conf-D	5362	14486	20515	8035	19516	16428	52968	53215

Tabla 2. Resultados (en milisegundos) de la ejecución de diferentes fractales.

Como se puede comprobar, las ejecuciones con la configuración-B son algo peores que con la versión centralizada (configuración-A). Este resultado era de esperar puesto que se invierte tiempo en la construcción de los mensajes, en su envío y en la recepción de las respuestas siendo que el tiempo para el cálculo permanece constante.

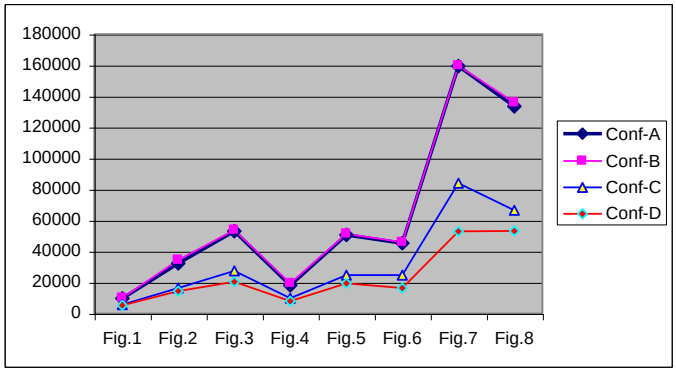

Figura 3. Comparativas de la distribución de cálculo.

En cuanto a las ganancias obtenidas al ubicar más de un servidor, en la tabla 3 se especifican en % los tiempos ahorrados por las configuraciones C y D. La figura 4 refleja de manera gráfica dichos ahorros de tiempos.

Tabla-3	**Fig.1**	**Fig.2**	**Fig.3**	**Fig.4**	**Fig.5**	**Fig.6**	**Fig.7**	**Fig.8**
Conf-C	41.87	49.18	48.26	45.40	51.12	45.50	47.33	50.12
Conf-D	45.01	55.23	61.37	55.77	61.43	63.76	66.76	60.12

Tabla 3. Ganancias (en %) por la presencia de varios servidores.

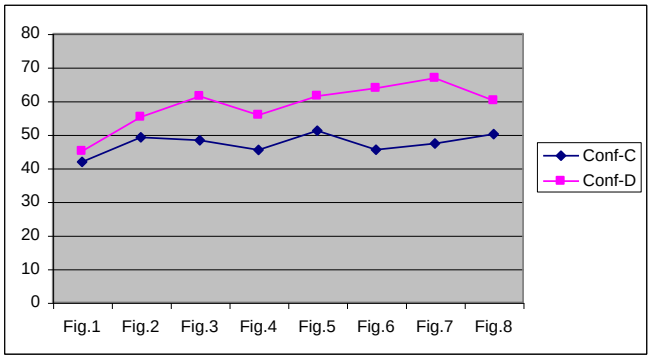

Figura 4. Ahorro de tiempos de cómputo con varios servidores.

Observando la figura 4, podemos ver que con dos servidores (Conf-C) obtenemos una reducción aproximada del 50% del tiempo. Sería de esperar que con tres servidores la reducción se aproximase al 75%. Sin embargo, esta reducción es algo menor. La explicación es que la distribución de carga entre los servidores no es uniforme; es decir, la carga de cada servidor viene impuesta por la zona del fractal que debe calcular. Puesto que diferentes zonas no conllevan la misma cantidad de cálculo y puesto que el tiempo total es el tiempo del servidor más lento, parece lógico concluir que raras veces nos aproximaremos a la ganancia teórica del 75%.

6. Conclusiones.

En este artículo hemos puesto de relieve que se pueden abordar problemas intensivos en cálculo sin tener que utilizar supercomputadores. La distribución del cálculo entre procesadores convencionales es una solución más barata y por tanto más atractiva para las organizaciones. El código presentado en la sección 3 pone de manifiesto que para abordar aplicaciones distribuidas es muy útil la presencia de un middelware (en este caso CORBA) que nos proporcione las operaciones necesarias sobre objetos distribuidos. Características básicas de las aplicaciones distribuidas como la compartición de recursos, la concurrencia o la transparencia, son servicios que ya proporcionan estos middelware. La escalabilidad del sistema es inmediata a partir de la inspección del código y de la arquitectura establecida. Si quisieramos hacer intervenir otro servidor basta con dar de alta en éste a la clase *ServidorMandelbrot* y a su FactoryFinder en el servicio de nombrado; desde el cliente se redistribuyen las filas a cada servidor y se crea un nuevo objeto servidor en el nuevo ordenador enviándole la petición de cálculo. Por último, con las medidas efectuadas sobre el cálculo de fractales, hemos puesto de relieve que la selección de más potencia de cálculo a base de utilizar más procesadores no asegura la disminución de los tiempos totales. Es necesario realizar una cuidadosa distribución de carga.

Referencias.

[1] Cincom. *VisualWorks® Distributed Smalltalk. Programmer`s Reference*. Cincinnati, Ohio: Cincom Systems Inc., 2000.

[2] Mandelbrot, B.B. *The Fractal Geometry of Nature*. :W.H. Freeman., 1983

[3] OMG. *CORBAservices: Common Object Services Specification*., 1998

[4] OMG. *The Common Object Request Broker: Architecture and Specification*., 1999

[5] Orfali, R., Harkey, D. y Edwards, J. *The Essential Distributed Objects, Survival Guide*. New York: John Wiley & Sons, Inc., 1996

[6] Rumbaugh, J., Jacobson, I. y Booch, G. *The Unified Modeling Language Reference Manual*. Reading, Massachussetts: Addison-Wesley., 1999

Apéndice-A.

A continuación se ilustran las zonas del plano complejo calculados junto con los parámetros para dichos cálculos y los resultados de las medidas realizadas.

F	Or-Real	Or-Imag	AnR	AlR	An	Al	N
1	-1.9d	1.0d	2.6d	1.95d	189	142	128
2	-0.88201058201058d	0.32711267605634d	0.325d	0.24375d	268	201	128
3	-0.76074192529416d	0.1355081984444d	0.040625d	0.03046875d	349	262	128
4	-0.74654063589588d	0.13306604672684d	0.005078125d	0.00380859375d	253	190	128
5	-0.7437105346113d	0.13214396613474d	6.34765625d-4	4.7607421875d-4	340	255	128
6	-0.7437011998227d	0.13187885813841d	1.26953125d-4	9.521484375d-5	306	230	128
7	-0.74365556307841d	0.13183539049235d	2.5390625d-5	1.904296875d-5	451	338	256
8	-0.74364711830292d	0.13182919307649d	8.4635416666667d-6	6.34765625d-6	272	204	1024

Or-Real:origen real; Or-Imag: origen imaginario; AnR: ancho del rectángulo complejo
AlR: alto del rectángulo complejo; An: ancho de la ventana de dibujo
Al: alto de la ventana de dibujo ; N: número de iteraciones

Tabla-1	**m-1**	**m-2**	**m-3**	**media**
Conf-A	9837	9702	9713	9751
Conf-B	10090	10018	10023	10044
Conf-C	5368	5820	5816	5668
Conf-D	5321	4998	5766	5362
Tabla-2	**m-1**	**m-2**	**m-3**	**media**
Conf-A	32031	32545	32483	32353
Conf-B	35325	33518	34419	34421
Conf-C	16461	16363	16494	16439
Conf-D	14775	14357	14326	14486
Tabla-3	**m-1**	**m-2**	**m-3**	**media**
Conf-A	52819	54276	52248	53114
Conf-B	53578	53577	54115	53757
Conf-C	27159	27433	27844	27479
Conf-D	20733	20372	20440	20515
Tabla-4	**m-1**	**m-2**	**m-3**	**media**
Conf-A	18057	18242	18207	18169
Conf-B	19263	19577	19578	19473
Conf-C	9652	9954	10155	9920
Conf-D	8278	7843	7983	8035

Tabla-5	**m-1**	**m-2**	**m-3**	**media**
Conf-A	49939	51152	50698	50596
Conf-B	50963	51090	51860	51304
Conf-C	24630	24409	25160	24733
Conf-D	19387	19302	19858	19516
Tabla-6	**m-1**	**m-2**	**m-3**	**media**
Conf-A	45291	44852	45835	45326
Conf-B	46131	45793	46043	45989
Conf-C	24887	24970	24252	24703
Conf-D	16384	16305	16594	16428
Tabla-7	**m-1**	**m-2**	**m-3**	**media**
Conf-A	159451	157930	160671	159351
Conf-B	158009	160576	160605	159730
Conf-C	86308	82531	82951	83930
Conf-D	53742	51808	53355	52968
Tabla-8	**m-1**	**m-2**	**m-3**	**media**
Conf-A	134172	130734	135443	133450
Conf-B	138434	134991	134303	135909
Conf-C	67417	65737	66523	66559
Conf-D	54394	51584	53667	53215

Figura 1

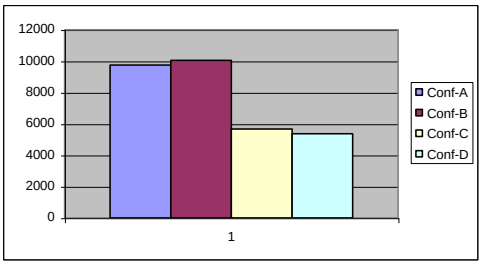

Gráfico 1

Figura 2

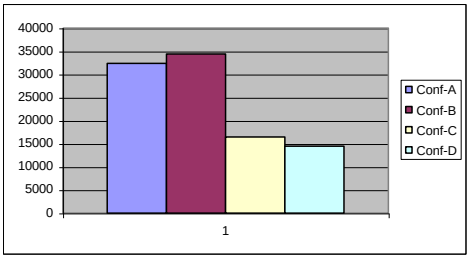

Gráfico 2

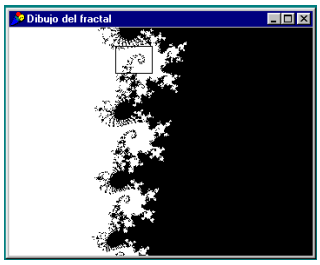

Figura 3

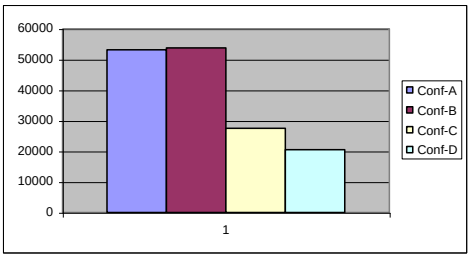

Gráfico 3

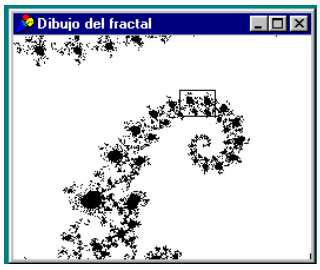

Figura 4

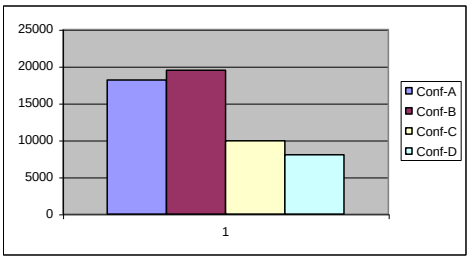

Gráfico 4

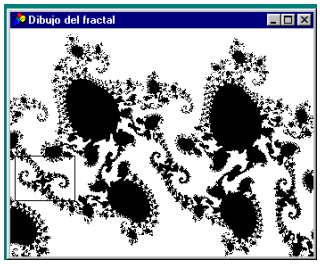

Figura 5

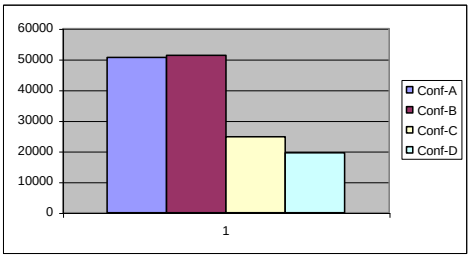

Gráfico 5

Figura 6

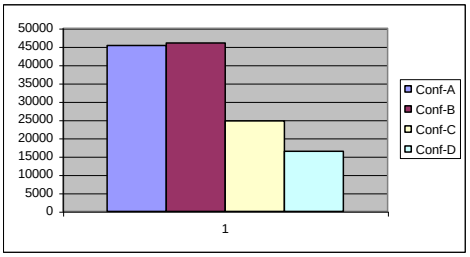

Gráfico 6

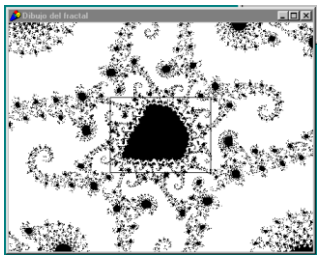

Figura 7

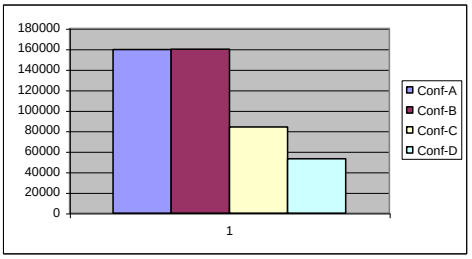

Gráfico 7

Figura 8

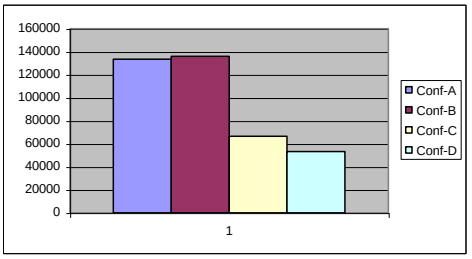

Gráfico 8