Paralelización de un Algoritmo de Compresión que Utiliza Diccionario Estático

Lic. H. Ramón[1], Lic. C.Russo[2], A.C. A. Anderson[3], A.C. D. Dirazar[3], Ing. A. De Giusti[4]

Laboratorio de Investigación y Desarrollo en Informática[5]
Departamento de Informática - Facultad de Ciencias Exactas
Universidad Nacional de La Plata

1. Resumen

Es notoria la importancia creciente de la compresión de datos en la transmisión de información en redes. Aspectos como seguridad, reducción de tráfico, optimización del uso del canal, etc., justifican las tareas de investigación y desarrollo en este tema, así como la evolución de los recursos tecnológicos disponibles [Russ95].

En particular, por el volumen de información en juego, en los últimos años hay un esfuerzo muy notorio en la optimización de algoritmos para compresión de datos utilizando diferentes tecnologías en software y en hardware que implican paralelización de métodos.

En este contexto, se presentan resultados experimentales de la comparación de implementaciones paralelas y lineales de un algoritmo de compresión de datos que utiliza diccionario, utilizando las computadoras de la red académica de la Universidad para la distribución y comunicación de procesos aplicando sockets.

[1]Prof. Adjunto con Ded. Excl. LIDI. Dpto. de Informática, Facultad de Ciencias Exactas, UNLP.
E-mail crusso@ada.info.unlp.edu.ar
[2]JTP Ded. Excl. LIDI. Dpto. de Informática, Facultad de Ciencias Exactas, UNLP.
E-mail hramon@ada.info.unlp.edu.ar
[3]Analista de Computación. Alumno de Lic. en Informática, Facultad de Ciencias Exactas, UNLP.
E-mail dirazar@ada.info.unlp.edu.ar.
[4] Inv. Principal CONICET. Profesor Tit. Ded. Excl., Dpto. de Informática, Facultad de Cs. Exactas, UNLP.
E-mail degiusti@ada.info.unlp.edu.ar
[5]Calle 50 y 115 Primer Piso, (1900) La Plata, Argentina, Teléfono 54-21-227707
E-mail lidi@ada.info.unlp.edu.ar

2. Introducción

Debido al crecimiento en el volumen de información que se utiliza en los últimos tiempos, se buscan métodos que faciliten el manejo y almacenamiento de esta, es por esto que temas como compresión de datos poseen interés [Nels91].

No solo nos preocupa el almacenamiento y el mantenimiento de los datos, sino también la disponibilidad en forma segura y rápida. El problema no acaba con optimizar el manejo local de los datos, debe resolverse también el problema de transmitirlos [Russ90].

Dentro de una red, el recurso compartido es el canal, de ahí, que la performance de la red dependa en gran parte del ancho de banda del canal. Si fuese posible disminuir la cantidad de información a transmitir, se lograría mayor fluidez en el tránsito del canal y por ende se mejoraría la performance global de la red.

Es por eso que debemos poner énfasis en analizar la conveniencia de minimizar el tráfico de datos por la red, problema especialmente importante cuando el medio de comunicación es costoso (en tiempo y/o dinero).

Las técnicas de compresión intentan reducir la cantidad de bits requeridos para almacenar o transmitir la información, sin pérdida o con pérdida, dependiendo del tipo de dato que se esta comprimiendo (voz, imágenes, textos, etc.).

Básicamente se buscan distintos métodos de compresión de información para [Russ95]:

- Reducción de tiempos en la transmisión de datos
- Menor requerimiento de memoria para almacenamiento ó

En la actualidad se han desarrollado distintos métodos de compresión con las siguientes propiedades:

- Tipo de método: con pérdida o sin pérdida de información
- Nivel de complejidad desde el punto de vista computacional
- Rango de datos sobre el cual el método puede ser utilizado obteniéndose resultados satisfactorios.

Se presenta el algoritmo de Huffman estático con el objeto de comparar las implementaciones sobre el paralelismo propuesto, se plantean dos modelos de solución y para cada uno de ellos se implementa en forma monoprocesador y multiprocesador.

Decidimos implementar en UNIX dado que nos manejábamos con herramientas de desarrollo standard como memoria compartida, semáforos y sockets, pensando luego en migrar hacia otro tipo de software y hardware (DSP's, Transputers, etc.) [Tine95].

3. Algoritmo de Huffman

La codificación de Huffman es probablemente el método más conocido de compresión de datos. La codificación de Huffman tiene aplicaciones prácticas: por ejemplo se utiliza en la última fase de compresión del método JPEG, en el método de compresión MNP-5 utilizado en los modems, etc.

Huffman trabaja de manera muy similar al código Morse, en la mayoría de los textos o imágenes, algunos símbolos y/o letras son más probables que otros. Esta observación sugiere un esquema de codificación en el cual a los símbolos más comunes se les asignen códigos cortos y a los símbolos menos frecuentes códigos largos.

La codificación de Huffman formaliza la idea de la longitud relativa de un símbolo de acuerdo a la probabilidad de ocurrencia del símbolo. La codificación estática requiere de una tabla con las probabilidades antes de comenzar la compresión de datos, esta puede ser generada de acuerdo a la fuente de entrada o contar con una fuente existente.

⇒ **PASO 1**: Escribir todos los símbolos con la probabilidad asociada de cada uno. El algoritmo consiste en la construcción de un árbol binario que tendrá a estos símbolos como nodos terminales. Inicialmente todos los nodos están sin marcar.

⇒ **PASO 2**: Encontrar los dos nodos con menor probabilidad y marcarlos. Agregar un nuevo nodo con un arco a cada uno de los nodos recién marcados. La probabilidad de este será la suma de las probabilidades de los dos nodos marcados anteriormente.

⇒ **PASO 3**: Repetir el paso 2 hasta que todos los nodos hayan sido marcados excepto uno. La probabilidad del nodo no marcado debe ser siempre uno.

⇒ **PASO 4**: La codificación de cada símbolo se hará trazando el camino desde el nodo no marcado al nodo terminal, teniendo en cuenta la secuencia tomada. El código asignado a cada símbolo es el camino seguido con izquierda=0 y derecha=1 o viceversa.

Este código tiene la propiedad prefija, esto significa que el código de cualquier elemento no es una subcadena inicial del código de otro elemento. Por lo tanto no existe ambigüedad al decodificar cualquier mensaje que usa el código de Huffman.

Para la descompresión, el receptor deberá conocer la tabla de frecuencias. Por lo tanto, la tabla deberá ser enviada en el paquete si fue armada en función de la fuente.

La tarea del receptor para descomprimir se reduce a una búsqueda en un árbol binario.

Se eligió paralelizar el algoritmo de Huffman semiestático por su simpleza, esto nos permitió abocarnos al problema de paralelización. Un refinamiento general de estos pasos corresponden a:

- **Comprimir**

```
Crear_histograma(Archivo_de_entrada, &Histograma);
Armar_arbol(Histograma, &Arbol);
Armar_codigo_y_header(Arbol, &Header, &Codigo);
Grabar_header(Archivo_de_salida, Header);
Codificar(Archivo_de_entrada, Codigo,Archivo_de_salida);
```

- **Descomprimir**

```
Rearmar_arbol(Archivo_de_entrada,&Arbol);
Decodificar(Archivo_de_entrada, Arbol, Archivo_de_salida);
```

El archivo comprimido posee la estructura física que muestra la figura:

4. Primer Modelo

En esta solución primeramente se arma la tabla de frecuencias de la fuente, luego se divide la fuente a comprimir en slots, de tamaño fijo, los cuales se comprimen con la tabla de frecuencias armada. Nótese que existe un único diccionario para la codificación de la fuente. Esta se implementó de dos maneras, una como monoprocesador y otra multiprocesador.

En las soluciones, por razones de eficiencia, en la compresión el código se generó como una tabla que se indexa por el símbolo y se obtiene la palabra de código correspondiente. En la decodificación, el código se regenera como un árbol.

4.1. Paralelo Monoprocesador

Esta implementación nos permite aprovechar los **tiempos muertos** de E/S, es decir, cada vez que un proceso realiza una E/S otro proceso accede a la CPU y realiza trabajo productivo.

Los procedimientos que se paralelizaron son:

- **En la compresión**

```
Crear_histograma(Archivo_de_entrada, &Histograma);
Codificar(Archivo_de_entrada, Codigo, Archivo_de_salida);
```

- **En la descompresión**

```
Decodificar(Archivo_de_entrada, Arbol, Archivo_de_salida);
```

La estructura física del archivo comprimido es:

Tamaño del Archivo Original
Tamaño del Slot
Código Comprimido - Inorder del ***"Arbol de Huffman"***

y la de los slots es:

Offset del slot descomprimido en el archivo original
Tamaño en bits del slot comprimido
Slot comprimido

4.2. Paralelo Multiprocesador

Se divide el archivo a comprimir en slots de igual tamaño. N procesos se comunican con los servers y requieren el servicio de generar el histograma del slot, acumulando esta información localmente. Al finalizar *suman sus histogramas locales* a un histograma global.

En este punto, otro proceso genera el código de Huffman correspondiente al histograma global, luego se disparan N procesos que se comunican con los servers utilizando el servicio de comprimir un slot. Los procedimientos que se paralelizaron y distribuyeron son:

- **En la etapa de compresión**

```
Crear_histograma(Archivo_de_entrada, &Histograma);
Codificar(Archivo_de_entrada, Codigo, Archivo_de_salida);
```

- **En la etapa de descompresión**

```
Decodificar(Archivo_de_entrada, Arbol, Archivo_de_salida);
```

La estructura del archivo comprimido es:

Tamaño del Archivo Original
Tamaño del Slot
Código Comprimido - Inorder del ***Arbol de Huffman***
Slots Comprimidos

y la de los slots es:

Offset del slot descomprimido en el archivo original
Tamaño en bits del slot comprimido
Slot comprimido

Las aplicaciones **servers** son las encargadas de atender los requerimientos de la aplicación principal. Existen 3 tipos básicos de servicios:

- Generar Histograma
- Comprimir
- Descomprimir

Existen también dos servicios especiales que son utilizados para inicializar el diccionario en sus dos formatos:

- Inicialización de la tabla utilizada por los procesos compresores.
- Inicialización del árbol utilizado por los procesos descompresores.

5. Segundo Modelo

En este caso no se utilizó una codificación global sino que cada slot fue codificado en forma independiente de los demás. Es decir, se generó un código para cada slot de acuerdo a la estructura probabilística del mismo, al utilizar un diccionario para cada slot, produce que el header de los mismos sea mas grande.

La ventaja de esta implementación es que la compresión es más localizada por lo que los datos propiamente dichos serán mejor comprimidos. La desventaja es el espacio que se *desperdicia* con los headers ya que ahora se necesitará un header completo por cada slot.

Debemos tener en cuenta que si los slot son muy pequeños el tamaño de los Header del slot van a expandir y no comprimir el archivo original. Si los slot son muy grandes van a perder localidad en la compresión.

5.1. Paralelo Monoprocesador

Esta implementación nos permite aprovechar los **tiempos muertos** como mencionamos anteriormente. Un proceso padre setea la cantidad de tareas (de acuerdo al tamaño del archivo y el tamaño de los slots), luego dispara M procesos hijos (M<=cantidad de tareas) que se encargan de realizar c/u de las tareas.

El archivo comprimido posee la siguiente estructura:

Tamaño del Archivo Original
Tamaño de Slot
Slots Comprimidos

y la estructura de los Slots Comprimidos:

Tamaño del árbol
Arbol (Inorder del código)
Offset del slot descomprimido en el archivo original
Tamaño en bits de los datos comprimidos
Datos comprimidos

5.2. Paralelo Multiprocesador

Se divide el archivo a comprimir en slots de igual tamaño. N procesos comprimen el archivo en forma concurrente. Las acciones que realiza cada uno de estos procesos son las siguientes:

- Tomar un slot.
- Generar el histograma del slot.
- Generar el código de Huffman para ese histograma.
- Codificar el slot.

Utilizando el método de paralelización descripto en el punto anterior trasladamos los procesos codificadores y decodificadores a otros procesadores.

La estructura de este algoritmo es similar a la estructura del algoritmo anterior, es decir se crean M subprocesos y c/u se encarga de realizar una tarea por vez, pero a diferencia del algoritmo anterior, estos subprocesos no realizan procesamiento de datos sino que se comunican con servidores que realizan el procesamiento (generalmente en otros procesadores). Esto nos permite procesar los slots asincrónicamente.

La estructura del archivo comprimido es:

Tamaño del Archivo Original
Tamaño del Slot
Slots Comprimidos

y la los Slots:

Offset del slot descomprimido en el archivo original
Tamaño en bytes del slot a distribuir
Slot a distribuir (sub-slot) o datos-excepción

Si ***Tamaño en bytes del slot a distribuir*** es cero los campos son:

Repeticiones del carácter
Carácter

Si ***Tamaño en bytes del slot a distribuir*** es distinto de cero la estructura del slot a distribuir es:

Tamaño del árbol
Arbol (inorder del código)
Tamaño de los datos comprimidos
Datos comprimidos

Las aplicaciones **servers** son las encargadas de atender los requerimientos de la aplicación principal Existen 2 tipos de servicios:

- Comprimir
- Descomprimir

6. Resultados Experimentales

La siguiente tabla nos muestra los tiempos en segundos de comprimir y descomprimir archivos binarios.

Tamaño (Mbytes)	Lineal (Segs)		Multiprocesador (Primer Modelo) (Segs)		Multiprocesador (Segundo Modelo) (Segs)	
	Comp.	Descomp.	Comp.	Descomp.	Comp.	Descomp.
1	15	14	10	9	8	8
2	29	28	20	16	15	15
3	43	42	27	23	25	21
4	56	55	37	31	30	28
5	70	68	46	37	38	34
6	82	85	55	46	44	44
7	97	98	66	55	51	50
8	112	113	75	63	67	60
9	130	130	87	74	73	67
10	145	143	93	82	77	74

Tabla 1

La siguiente tabla nos muestra los tamaños y radio de compresión de los archivos binarios comprimidos y descomprimidos.

Archivo Original (Bytes)	Lineal (Bytes)		Multiprocesador (Primer Modelo) (Bytes)		Multiprocesador (Segunda Versión) (Bytes)	
	Tamaño	Radio	Tamaño	Radio	Tamaño	Radio
1048576	808461	0.771	809336	0.771	751774	0.716
2097152	1613977	0.769	1615720	0.770	1538862	0.733
3145728	2423129	0.77	2425738	0.771	2352460	0.747
4194304	3207835	0.764	3211306	0.765	3086357	0.735
5242880	3987564	0.741	3991884	0.761	3819914	0.728
6291456	4672048	0.742	4677218	0.743	4404216	0.700
7340032	5439204	0.741	5445243	0.741	5173208	0.704
8388608	6283910	0.749	6290814	0.749	6028895	0.718
9437184	7131528	0.755	7139304	0.756	6870884	0.728
10485760	7971162	0.760	7979797	0.761	7693116	0.733

Tabla 2

6.1. Soluciones multiprocesador

Las características de las distintas computadoras con LINUX es:

- ALGOL: 486DX2, 16 Mb. RAM
- ADA: 484DX2, 16 Mb. RAM
- ISIS: 486DX2, 32 Mb. RAM
- NAHUEL: 486DX2, 8 Mb RAM.

Procesadores	Primer Modelo Archivo 7 Mb		Segundo Modelo Archivo 5 Mb		Hosts Utilizados
	Comp.	Descomp.	Comp.	Descomp.	
1	98	85	74	63	ALGOL
2	72	68	48	44	ALGOL + ADA
3	63	61	38	34	ALGOL + ADA + ISIS
4	55	53	31	29	ALGOL + ADA + ISIS + NAHUEL

Tabla 3

7. Conclusiones

En cuanto a los tiempos y radios de compresión de los distintos modelos podemos decir que:

- Durante las pruebas realizadas, comparado los tiempos del Algoritmo lineal vs. Algoritmo Paralelizado Monoprocesador (primer modelo), en los mejores casos el paralelizado es igual al lineal, pero en la mayoría era peor. En los casos en que era igual, se estaba explotando los tiempos muertos de E/S. Con esto concluimos que, por un lado se alcanzaba el objetivo por el cual se había paralelizado (Uso de CPU en tiempos de E/S), pero, por otro, el tiempo que se ganaba con este uso intensivo de la CPU no compensaba el que se ***perdía*** con el manejo de procesos.
- Analizando la *Tabla 1* puede verse que el algoritmo distribuido primer modelo fue aproximadamente un 30% más rápido que el lineal. Analizando la *Tabla 2* vemos que los radios de compresión son similares, los que nos lleva a concluir que las mejoras de este modelo multiprocesador radica en los tiempos de compresión/descompresión.

- Analizando los dos modelos multiprocesador en la *Tabla 1* vemos que el segundo modelo es aproximadamente un 48% mas rápido que el primero, en la *Tabla 2* observamos que tenemos mejor radio de compresión en el segundo modelo que en el primero, debido la localidad de la codificación, podemos concluir que el segundo modelo posee mejor radio de compresión y tiempos que el primero.
- En la *Tabla 3* puede verse que si tenemos procesadores disponibles y para comprimir varios Mbytes las mejoras son importantes en tiempo, además el multiprocesador segundo modelo sigue superando al del primer modelo.

8. Trabajos Futuros

Se podría buscar el tamaño del para los slots de la segunda solución para optimizar el uso de la información local.

Este tipo de algoritmos se podría implementar sobre hardware o simuladores Multi-DSP [Tine95] en donde poseemos paralelismo a nivel de procesos y por la arquitectura Harvard, a nivel de instrucciones, además de los Transputers en donde podemos implementar en OCCAM.

Podemos además diversificar las aplicaciones, no dedicarnos solo a texto, sino Imágenes, voz, etc.

9. Bibliografía

[Andr91] *Concurrent Programming*. Gregory R. Andrews. The Benjamin Cummings Publishing Company Inc.

[Elora94] *Compresión de Imágenes en Dos Tonos*. Boraccia Marcos, Mas Carlos, Rodríguez Leandro. Trabajo de Grado, UNLP, 1994.

[Burn93] *Concurrent Programming*. Alan Burns, Geoff Davies. Addison Wesley.

[Chan88] *Parallel Program Desing*. K. Mani Chandy, Jayadev Misra. Addison Wesley.

[Code92] *Introduction to Parallel Processing*. Bruno Codenotti, Mauro Leoncini. Addison Wesley.

[Geha89] *Concurrent Programming*. Narain Gehani, Andrews D. McGettrick. Addison Wesley.

[Jank91] *Recursive Images*. Steven Janke, Dr. Dobb's, July 1991.

[Nels91] *The Data Compresión Book*. Mark Nelson, Prentice Hall, 1991.

[Rait93] *Image Compression with Affine Transformation*. Harri Raittinen, Tampere University Report 6/93.

[Ramo94] *Procesamiento Paralelo: Experiencias con Transputers y DSP*. Hugo Ramón, Claudia Russo, Fernando Tinetti, Marcelo Naiouf, Armando De Giusti, First International Congress of Information Engeneering, Bs. As., 1994.

[Russ90] *Combinación de Algoritmos de Criptografiado y Compresión de Datos en Redes de Procesadores*. Claudia Russo, Gabriela Rosanova, Armando De Giusti, 2do Congreso de Informática y Telecomunicaciones de la Provincia de Buenos Aires, CINTEBA 90, 1990.

[Russ95] *Paralelización de Algoritmos de Compresión Fractal de Imágenes*. Claudia Russo, Hugo Ramón, Marcos Boraccia, Second International Congress of Information Engeneering, Bs. As., 1995.

[Tine95] Análisis *y Extensión de una Herramienta de Simulación Multi-DSP para Prcesamiento Paralelo*. Tinetti, F., Ramon H., Russo, C., Second International Congress of Information Engeneering, Bs. As., 1995.