Paralelización del codificador H.264 con estimación de movimiento adaptativa en clusters de PCs[1]

A. Rodríguez[*], M. Pérez[*], A. González[*], J. Peinado[*], J.C. Fernández[+]
[*]Universidad Politecnica de Valencia, Dept. de Informática de Sistemas y Computadores
[+]Universidad Jaume I, Dpt. d'Enginyeria i Ciència dels Computadors

Resumen

La codificación de vídeo es de primordial importancia para la mayoría de aplicaciones multimedia, en especial las que manejan flujos de vídeo de alta calidad. El más reciente de los estándares de codificación de vídeo es el H.264 AVC, el cual ha incorporado novedosas mejoras que lo hacen especialmente eficiente para un rango amplio de aplicaciones. En particular, el H.264 AVC utiliza el esquema clásico basado en tratamiento de la señal y la codificación de residuos con compensación de movimiento. La novedad es el incremento de complejidad de los bloques operativos y por tanto el incremento considerable de computación. La utilización de clusters tiene como objetivo la reducción del tiempo de codificación con niveles de latencia que permitan a aplicaciones como la videoconferencia o la difusión de vídeo codificado trabajar con formatos de alta resolución y tasa de cuadros (HDTV, D-Cinema, ec.). En el presente trabajo se presenta un estudio sobre la paralelización del codificador H.264, proponiendo un mecanismo para realizar la estimación de movimiento de forma adaptativa que aplique toda la potencia de estimación del codificador sólo cuando sea realmente necesario.

1. Introducción

Actualmente hay múltiples aplicaciones en las que la codificación de vídeo es necesaria, algunos ejemplos son los teléfonos móviles de última generación, la difusión de vídeo en Internet y la televisión de alta definición. La información de vídeo se caracteriza por requerir grandes cantidades de bits con los consiguientes requerimientos de ancho de banda y de espacio para su transmisión y almacenamiento. Los codificadores de vídeo reducen drásticamente estos requerimientos introduciendo una cierta perdida de calidad en la señal de vídeo reconstruida. Aplicando las técnicas de compresión de forma adecuada dicha perdida puede controlarse de forma que sea aceptable en el contexto de la aplicación.

Los primeros codificadores de vídeo como MPEG [1][2] se basaron en las técnicas digitales de procesamiento de la señal (transformación, cuantificación y codificación de entropía). Actualmente, codificadores como MPEG-4 [9] utilizan otras técnicas más novedosas de segmentación y codificación de objetos.

El codificador H.264 AVC [3] (MPEG-4 parte 10) sigue el modelo clásico pero llevando al límite la sofisticación de cada uno de los elementos. Esto se traduce en una mayor efectividad en la compresión, o sea mayores tasas de compresión, y menor error perceptual, pero con unas exigencias de computación mucho mayores.

Con respecto a esto último, se sabe que una de las etapas más costosas del codificador H.264 es la estimación de movimiento que se lleva a cabo para cada uno de los cuadros de vídeo que se codifican como INTER (más de un 90% de los cuadros de una secuencia de vídeo). En la configuración del codificador H.264 se puede determinar el nivel de precisión con el que se quiere buscar el movimiento resultante en un cuadro. Por ello, pensamos que en determinadas situaciones no es necesario emplear demasiados ciclos de CPU para codificar el movimiento de una forma tan precisa, asumiendo la posible

[1] Este trabajo ha sido financiado por el proyecto de la Generalitat Valenciana GV04B-507

Actas de las XVI Jornadas de Paralelismo, JP2005, pp.591-597
ISBN: 84-9732-430-7

pérdida de calidad y eficiencia. Por todo ello, planteamos un esquema de estimación de movimiento adaptativa a nivel de grupo de cuadros (GOP) en donde se busca un compromiso entre la perdida de rendimiento del codificador y la reducción del tiempo de codificación.

Por otro lado, la posibilidad de disponer de entornos de alta productividad y la codificación en tiempo real en los contextos de aplicación más exigentes como la televisión de alta definición, obliga a utilizar plataformas con gran potencia de cálculo. Una alternativa posible son los clusters que ofrecen prestaciones escalables a bajo coste y con una gran flexibilidad.

El objetivo del presente trabajo es conseguir implementaciones del codificador H.264 AVC sobre clusters que sean escalables y eficientes, de forma que se puedan encontrar configuraciones de cluster adecuadas para las exigencias de las aplicaciones de codificación de vídeo más extremas.

La organización del presente trabajo es la siguiente: En la sección 2 planteamos el mecanismo de estimación de movimiento adaptativa, justificando su interés. En la sección 3, se presentan los esquemas de paralelización a nivel de GOPs. En la sección 4, mostramos algunos resultados de rendimiento de las versiones propuestas y, finalmente, en la sección 5 exponemos las conclusiones del trabajo y planteamos algunas acciones que se pretenden llevar cabo en esta línea.

2. Estimación de movimiento adaptativa

Con el objeto de reducir el coste computacional de la estimación de movimiento del codificador H.264 AVC, se propone realizar la estimación de movimiento de forma adaptativa en base a una función de coste discreta que maximice la relación entre el rendimiento del codificador (calidad y tasa de bits) y el tiempo de codificación. Esta función de coste deberá determinar que nivel de precisión tiene que emplear el codificador para identificar el movimiento en el GOP actual.

A priori, el codificador H.264 permite definir la precisión del mecanismo de estimación de movimiento a través de un conjunto de parámetros de configuración. Ajustando el valor de dichos parámetros podemos definir varios niveles de precisión. En particular, y para este trabajo, hemos definido tres niveles diferentes. El primero define la máxima precisión posible (opción por defecto del codificador, le llamamos E0), el segundo nivel ofrece una precisión intermedia usando una búsqueda con tamaños de bloque de 16x16, 16x8, 8x16 y 8x8 (le llamamos E1), mientras que el tercer nivel sólo utiliza el tamaño de bloque de 16x16 (le llamamos E2).

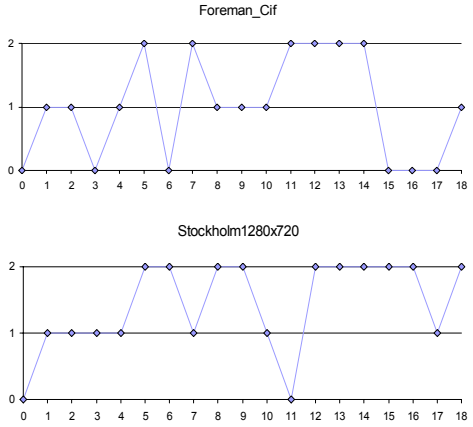


Figura 1. Selección adaptativa de los niveles de estimación de movimiento (E0,E1 y E2).

En la figura 1, podemos observar el nivel de precisión seleccionado para cada GOP en las secuencias Foreman (CIF@30fps) y Stockholm (HDTV@30fps). Esto se ha realizado de forma manual simulando una función de coste de tres niveles para cada GOP (a este esquema lo llamamos EA). Lo anterior da como resultado importantes reducciones del coste temporal del algoritmo, con respecto a la codificación estándar con estimación de movimiento exhaustiva (E0).

Con el objeto de observar cuanto tiempo de cómputo puede recortar la estimación de movimiento adaptativa, realizamos un experimento codificando las dos secuencias de vídeo mencionadas anteriormente, con la versión secuencial JM 7.6 [11] del codec H.264 AVC.

En la figura 2, se muestra la reducción del coste temporal de los GOPs codificados con los niveles de estimación definidos: E1, E2 y EA. La función de coste de EA busca reducir al mínimo el tiempo de codificación, manteniendo al máximo la calidad, sin incrementar demasiado el tamaño

del bitstream, como se puede observar en las figuras 3 y 4.

En el esquema adaptativo (EA) la pérdida de calidad provocada corresponde a centésimas de decibelio. Esta pérdida de calidad la podemos considerar despreciable, siendo este un factor prioritario a la hora de codificar video de alta calidad.

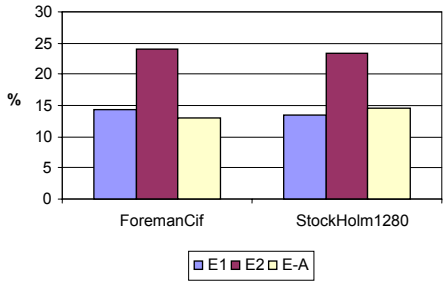


Figura 2. Porcentaje de reducción del tiempo de codificación respecto de E0.

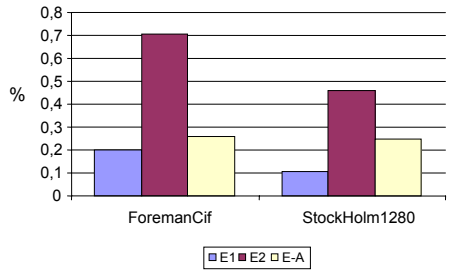


Figura 3. Porcentaje de reducción del PSNR

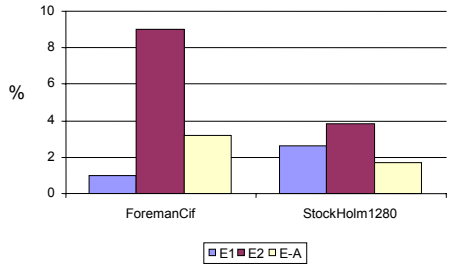


Figura 4. Porcentaje de incremento del bitrate respecto de E0.

En cuanto al tamaño del bitstream generado, se observa un incremento medio que ronda el 3% para la estimación de movimiento adaptativa (EA). Este incremento de tamaño es menos apreciable en secuencias de alta resolución (secuencia Stockholm), siendo este el precio que asumimos como contrapartida cuando empleamos estimación de movimiento adaptativa.

Por tanto, podemos decir que utilizar un esquema de estimación de movimiento adaptativo es más que interesante ya que la calidad perceptual del video decodificado no se ve alterada, mientras que incrementamos ligeramente el tamaño del bistream. Este incremento del bitrate no será determinante para un gran abanico de aplicaciones (difusión de TV digital en formatos HDTV, videoconferencia de alta calidad, etc.), en las que el retardo del codificador es un parámetro crítico y se debe mantener la calidad de la señal al máximo.

3. Esquemas de paralelización

En la literatura se pueden encontrar múltiples versiones paralelas de codificadores de video, particularmente MPEG-1[2][5][8] y MPEG-2 [1][6]. Las implementaciones paralelas del codificador H.264 que presentamos se han basado en el código JM 7.6 disponible de forma libre [11] utilizando MPI [7].

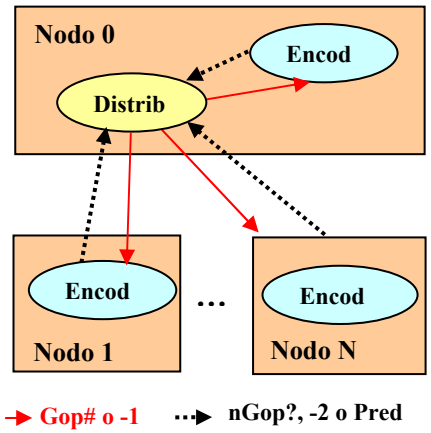


Figura 5. Esquema de distribución por GOPs y tipos de mensajes

El paralelismo se consigue descomponiendo la secuencia de vídeo en bloques de 15 cuadros consecutivos (GOPs), de forma que cada bloque sigue un patrón de codificación del tipo IBBPBBP…PBBI [10]. La codificación de cada GOP es una tarea totalmente independiente, por lo que los nodos no requieren comunicación alguna con los demás para codificar su GOP.

El video a codificar se encuentra en un sistema de ficheros compartido, al que todos los nodos pueden acceder para leer los GOPs. De la misma forma, el bistream resultante de codificar cada GOP es almacenado en ficheros adecuadamente identificados para el ensamblado final en un único bitstream. Este proceso se realiza cuando el último nodo ha terminado de codificar su GOP.

En la figura 5 se muestra un diagrama que ilustra el funcionamiento del codificador paralelo con descomposición de datos a nivel de GOP. En la parte superior se encuentra el proceso 0 que inicia el lanzamiento del programa y posteriormente ejecuta dos subprocesos: distribuidor (Distrib) y codificador (Encod).

Los mensajes que intercambia el proceso distribuidor con los procesos de codificación son los siguientes:

- **n_Gop?: T**erminación del proceso de codificación del GOP actual y solicitud de un nuevo GOP.
- **Gop#: I**ndica el numero de GOP a codificar.
- **-1:** No hay más GOPs a codificar.
- **Pred:** Corresponde con el informe de predicción del tiempo estimado para finalizar el GOP en curso del proceso codificador. Este valor solo es usado por el esquema de distribución por estimación de coste.
- **-2:** Confirma la terminación de la codificación en un determinado nodo. Este mensaje es necesario porque al final se tienen que integrar todos los GOPs codificados en un solo fichero de salida.

Experimentalmente hemos observado que el tiempo de codificación de los GOPs tiene una variación muy pequeña, de forma que las características de la secuencia de vídeo no parecen condicionar mucho el tiempo de codificación. Esto es así en el codificador original, lógicamente se pueden aplicar optimizaciones que hagan que la cantidad de cómputo, en particular en la estimación y compensación de movimiento, se adapte a las características de la secuencia en cada GOP (nivel de detalle, contornos, cantidad de movimiento, etc).

La escasa variación del tiempo de codificación a nivel de GOP sugiere que no será complicado obtener algoritmos paralelos con buen equilibrio de la carga, que utilicen la codificación de un GOP como tarea básica.

Sin embargo, al utilizar el mecanismo de estimación de movimiento adaptativa, se introduce un desbalanceo en la carga del codificador, haciendo que existan diferencias de coste computacional significativas entre los GOPs de una secuencia de vídeo. En este caso, el esquema de asignación de GOPs a procesadores puede jugar un papel importante en cuanto al rendimiento del codificador paralelo se refiere.

Para poder estudiar este fenómeno, proponemos utilizar dos esquemas de asignación de GOPs: asignación por demanda y asignación por estimación de coste.

3.1 Asignación por demanda.

Este esquema asigna de forma consecutiva un GOP a cada nodo en la primera ronda de asignación. Cuando un nodo finaliza la codificación del GOP actual, lo indica al proceso distribuidor solicitando la asignación de un nuevo GOP (ver figura 5). El proceso distribuidor atiende la petición del nodo asignándole el primer GOP pendiente de codificar. Este proceso continúa hasta que no quedan más GOPs, en cuyo caso el proceso distribuidor notifica esta situación a los nodos para que finalicen la ejecución.

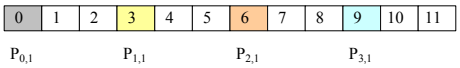
La asignación por demanda va procesando los GOPs en el orden de aparición en el bistream a codificar. A continuación se muestra el algoritmo del proceso distribuidor para la asignación por demanda:

- **Asigna a los procesadores su primer GOP**
- **Repetir hasta fin de secuencia**
  - **Espera mensaje n_Gop?**
  - **Envía Gop# del siguiente GOP al procesador**
- **Para cada procesador**
  - **Espera mensaje n_Gop?**
  - **Envía -1 al procesador**
- **Ensambla los GOPs codificados en un fichero único**

3.2 Asignación por estimación de coste.

Cuando utilizamos la asignación por estimación de coste se trata de asignar GOPs de forma que se tenga en cuenta el balanceo de carga que el sistema lleva hasta el momento. Además se tendrá en cuenta la correlación del coste temporal en los frames/GOPs consecutivos.

Para ello se propone cubrir diferentes zonas de la secuencia de vídeo. Así, la asignación inicial de GOPs a procesadores estaría distribuida a lo largo de toda la secuencia.



Supongamos una secuencia de 12 GOPs para cuatro nodos. Al nodo 0 se le asigna el primer GOP, al nodo 1 el GOP #3 y así sucesivamente.

El proceso distribuidor deberá recoger informes de previsión de coste de los GOPs que se están codificando en todos los nodos, con el objeto de hacer una planificación que intente mejorar dinámicamente el balanceo de carga de todos los procesadores.

Cuando los nodos, en la codificación del GOP asignado, disponen de una buena predicción[2], envían al proceso distribuidor un informe de estimación de coste, indicando cuanto tiempo les falta para terminar de codificar el GOP en curso.

El proceso distribuidor recoge los informes de los nodos antes de que finalice el GOP más ligero y realiza una planificación de asignación, de forma que el GOP previsiblemente más pesado (el vecino del GOP más pesado de los que se están procesando actualmente) se asigne al nodo que ha procesado el GOP mas ligero y viceversa.

Siguiendo esta política, tratamos de distribuir los GOPs entre los procesadores, en función de las previsiones de coste de los que se están codificando actualmente, suponiendo que si un GOP es ligero el vecino también lo será. A continuación se muestra el pseudo-código de este mecanismo para el proceso distribuidor:

```
• Asigna a cada procesador su primer GOP (ver sección 3.2)
• Calcula el número de ciclos de asignación de GOPs (Nciclos)
• Para cada ciclo C = 1..Nciclos-1
  ○ Para cada procesador P = 0..Nproc-1
    ○ Espera mensaje Pred (predicción de P)
    ○ Si es la primera predicción recibida lanza timer (tout = prediccion)
    ○ Si no Si predicción < tout relanza el timer (tout = prediccion)
  ○ Si el timer no ha vencido
    ○ Detener el timer
    ○ Lanzar planificación (ver sección 3.2)
• Para cada procesador P
  ○ Espera mensaje Pred (el último)
  ○ Envía -1 (no hay mas GOPs)
  ○ Espera -2 (fin de proceso de P)
• Ensamblar GOPs codificados en fichero único
```

El pseudo-código del manejador del timer es el siguiente:

```
• Lanza planificación (ver sección 3.2)
• Para cada procesador P = 0..Nproc-1
  ○ Envía mensaje Gop# con el ID de GOP asignado
```

4. Resultados obtenidos

La evaluación de los algoritmos propuestos se ha realizado sobre dos secuencias de vídeo: Foreman (CIF) y Stockholm (HDTV). Los experimentos se han realizado sobre un cluster compuesto por cuatro nodos biprocesadores con Opteron 246 a 2GHz, con 1 GByte de memoria por nodo, interconectados por una red Gigabit Ethernet con agregación de dos enlaces por nodo y un switch 3Com. El sistema operativo es Linux SuSE 9.1 y el entorno MPI es MPICH-2. Los codificadores paralelos se basan en dos esquemas de asignación de GOPs y en dos esquemas de estimación de movimiento:

- *Esquemas de asignación de GOPs:*
  - Asignación por demanda.
  - Asignación por estimación de coste.
- *Esquemas de estimación de movimiento:*
  - Estimación exhaustiva
  - Estimación adaptativa

Los tiempos de referencia son los obtenidos por el codificador con asignación por demanda y estimación de movimiento exhaustiva. Las figuras 6 y 7 muestran el porcentaje de reducción del tiempo de codificación de los codificadores paralelos con estimación de movimiento adaptativa, en el caso de asignación de GOPs por demanda y por estimación de coste, sobre 24 GOPs de las dos secuencias de vídeo mencionadas.

[2] Experimentalmente se han obtenido buenas estimaciones cuando se ha procesado el 80% del GOP.

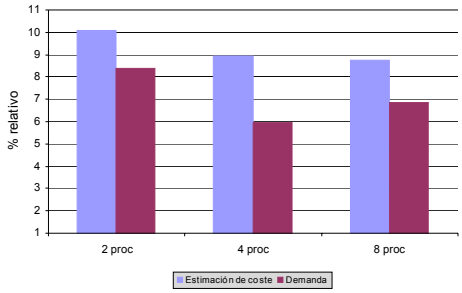


Figura 6. Reducción del tiempo de ejecución para la secuencia Stockholm

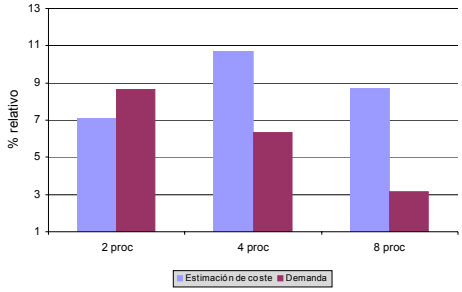


Figura 7. Reducción del tiempo de ejecución para la secuencia Foreman

Los resultados obtenidos muestran como los codificadores H264 paralelos, con estimación de movimiento adaptativa, reducen el tiempo de codificación. En el caso de la asignación de GOPs por estimación de coste las mejoras están entorno a un 10%, siempre por encima de la asignación por demanda cuando se tienen menos de 8 GOPs por procesador.

Los speedups para los codificadores paralelos basados en estimación de movimiento adaptativa se muestran en las figuras 8 y 9. Se observa una mejora para el caso de la asignación de GOPs por estimación de coste, tal y como se esperaba. Aunque la magnitud de la mejora, en las secuencias consideradas, es relativamente pequeña.

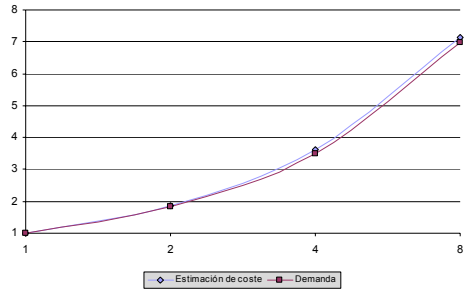


Figura 8. Speed-ups para la estimación adaptativa para la secuencia Stockholm

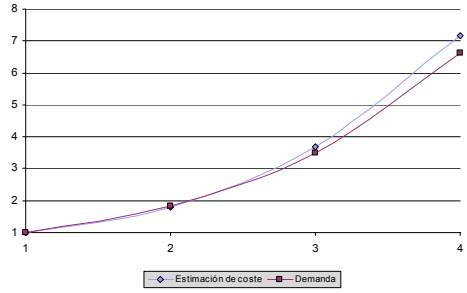


Figura 9. Speed-ups para la estimación adaptativa para la secuencia Foreman

5. Conclusiones y trabajos futuros

Se han obtenido mejoras apreciables con los algoritmos propuestos que combinan la estimación de movimiento adaptativa y el control del equilibrio de la carga basada en la estimación de coste, todo ello a nivel de GOP. Es de suponer que este esquema dará mejores resultados al introducir optimizaciones en el codificador que den lugar a una mayor variación del tiempo de codificación en función de las características de cada GOP.

Los esquemas propuestos, en particular la asignación por estimación de coste, no serán adecuados si la latencia es un parámetro de importancia ya que necesitan que una buena parte de la secuencia de vídeo esté disponible. Sin embargo, sí serán adecuados para la codificación de video almacenado o para aplicaciones en las que el valor de latencia admisible sea elevado.

Como trabajos futuros nos planteamos dos líneas de trabajo complementarias:

- La evaluación del esquema de asignación por estimación de coste introduciendo nuevas optimizaciones.
- La paralelización de la codificación de cada GOP a varios niveles:

1. Descomposición en subcuadros.
2. Paralelización de los bucles de cómputo más intensivos.
3. Paralelización SIMD de las operaciones replicadas.

Referencias

[1] Akramullah, S.M., Ahmad, I., Liou, M.L.: Performance of a Software-Based MPEG-2 Video Encoder on Parallel and Data Distributed Computing Systems. IEEE Transactions on Circuits and Systems for Video Technology, vol. 7. (1997) 687-695

[2] Gong, K.L.: Parallel MPEG-1 video encoding. MS thesis, Department of EECS, University of California at Berkeley, Issued as Technical Report, May (1994)

[3] ITU-T: H.26L Test Model Long Project (TML-8), July (2001)

[4] Kralevich, N.: The NOW Split-C MPEG Encoder. Internal Report Final Project CS267, University of California at Berkeley. Issued as Technical Report

[5] Martínez, L.F.: An Efficient ISO/MPEG-1 Layer II Encoder using Parallel Processing Techniques. Proyecto de investigaci\'on de la Universidad de Miami, Florida (1995)

[6] Olivares, T., Quiles, F., Cuenca, P., Orozco-Barbosa, L., Ahmad, I.: Study of Data Distribution Techniques for the Implementation of an MPEG-2 Video Encoder. Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems, MIT Boston (1999)

[7] Pacheco, P.S.: Parallel Programming with MPI, Morgan Kaufman Publishers, Inc

[8] Shen, K., Rowe, L.A., Delp, E.J.: A Parallel Implementation of an MPEG-1 Encoder: Faster then Real Time!. Conference on Digital Video Compression on Personal Computers: Algorithms and Technologies, San Jose-California (1995) 407-418

[9] A.Hamosfakidis, Y.Paker, J. Cosmas "*A study of Concurrency in MPEG-4 Video Encoder*", ICMCS' 98, Austin, Texas, USA. June 1998

[10] A. Rodriguez, A. González, M.P. Malumbres.: Performance evaluation of parallel MPEG-4 video coding algorithms on clusters of workstations. Parelec 04.

[11] http://iphome.hhi.de/suehring/tml/