H.264 解碼複雜度分析及其在 Libav 之實現

陳駿傑*[a]、楊士萱[a]
國立臺北科技大學資訊工程系[a]

摘要 — 本論文分析與預估開放軟體 Libav 解碼 H.264 影片的複雜度，以做為調整 CPU 電壓和頻率的依據，進而達到讓影片播放順暢與節省電量消耗的目的。我們將 libav 解碼器 libavcodec 解碼 H.264 的程式碼分成 6 個解碼單元(Decoding Module，DM)，分別量測各個 DM 運算的時間，並觀察運算時間與視訊編碼參數間的關係，以做為預測解碼複雜度的依據。我們測試了 5 個具有不同特性的視訊序列，比較實際測量到的時間和我們預估的時間，結果發現時間估測的誤差百分比平均僅為 3.4%、而標準差為 4.3%。[1]

一、 簡介

H.264/AVC (Advanced Video Coding)[1]視訊壓縮標準近年已經被廣泛使用於各種視訊應用，包括數位電視、行動視訊、視訊串流、與藍光碟片等。與先前的 MPEG-2 標準相比，H.264 用一半的碼率就可以得到相近品質的視訊。有效的 H.264 解碼流程與軟硬體實現，乃成為各項多媒體服務的關鍵技術。

現在比較進步的嵌入式平台，都可以動態調整 CPU 的電壓和頻率(Dynamic Voltage and Frequency Scaling，DVFS)，以減少電量的消耗。如果在播放影片時可以根據解碼的複雜度進行 DVFS，就可以讓播放影片所消耗的電量降低，延長電池壽命，也可以讓影片播放得更順暢。目前常見的 VLC 和 MPlayer 等媒體播放器，都使用開放原始碼的 FFmpeg 及內附之 libav 解碼器 libavcodec [2]做為主要內置解碼引擎，因此本論文以分析 libavcodec 解碼 H.264 影片的複雜度做為主要目標。

我們參考文獻[3]的做法，將 H.264 解碼器依照解碼流程分成 6 個解碼單元(decoding module、DM)。值得注意的是，文獻[3]使用自己開發的 H.264 解碼器進行複雜度預估，並非使用常用的開放原始碼。[4]探討了 H.264/AVC Baseline 解碼的複雜度，他們找出幾個較複雜的運算。[5]、[6]分別更進一步分析了熵解碼和動作補償的複雜度。[7]使用了統計的架構來預測解碼的複雜度，但跟[3]比較起來，其預測的方式相對複雜。[8]分析了 FFmpeg 解碼器解 H.264 影片的複雜度，他們利用了畫面解析度、每秒畫面張數、量化參數 QP (quantization parameter) 3 項編碼參數來預估，但卻不夠精細。

我們參考[3]的方法，將 libavcodec H.264 解碼分成 6 個 DM，分別為熵解碼(Entropy Decoding)、離散餘弦反轉換(Inverse Discrete Cosine Transform、IDCT)、畫面內預測 (Intra Prediction) 、動作補償 (Motion Compensation)、去區塊濾波(Deblocking Filtering)、以及準備附帶資訊(Side Information Preparation)等。我們在編碼時紀錄每個 DM 執行的次數 n_{DM}，再透過預測得到每個 DM 執行一次的複雜度 k_{DM}，便能利用這兩項參數估算出每個 DM 解碼的複雜度 C_{DM}。

本論文後續編排如下。第二節介紹測量複雜度的方法與軟體的設定，第三節介紹實驗結果，第四節則為結論。

[1] 本研究由國科會贊助，計畫編號 NSC 102-2219-E-027-002。

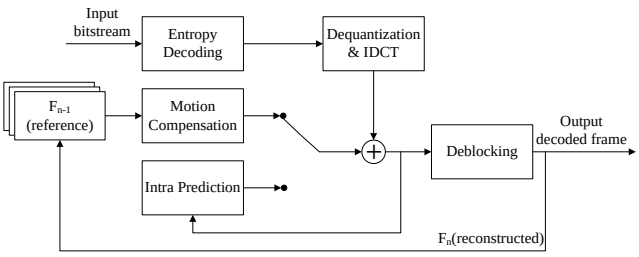
二、 H.264 解碼複雜度分析

2.1 解碼單元

我們參考[3]的方法將 libavcodec H.264 解碼端的程式碼區分成 6 個 DM，如圖一的 H.264 解碼流程圖所示。每個 DM 運算的時間 C_{DM}，我們可以把它看成 DM 執行的次數 n_{DM} 和 DM 執行一次複雜度 k_{DM} 的乘積：

$$C_{DM} = n_{DM} \cdot k_{DM} \quad (1)$$

n_{DM} 的資訊在編碼時便可以取得，而 k_{DM} 則需要進行預測，所以我們先觀察各種序列在每個 DM 實際運算的複雜度(以 $\hat{k}_{DM}$ 表示)的特性，再利用觀察到的特性來幫助我們預測 k_{DM}。



圖一： H.264 解碼流程圖

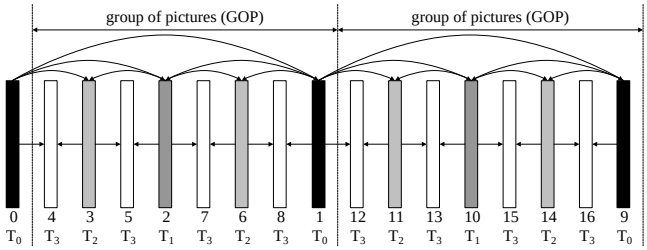
我們量測每張畫面的每個 DM 實際運算的時間 $\hat{C}_{DM}$ 除上對應的 n_{DM}，就可以得到 $\hat{k}_{DM}$：

$$\hat{k}_{DM} = \frac{\hat{C}_{DM}}{n_{DM}} \quad (2)$$

某些 DM 的 $\hat{k}_{DM}$ 變化很穩定可以設為常數，而有一些 DM 的 $\hat{k}_{DM}$ 需要依照畫面的類型(frame type)和 QP 來分類，譬如在圖二所示的階層式 B 畫面(Hierarchical B)的預測編碼架構中，T_0 和 T_1 兩個時間分層(temporal layer)上的畫面做畫面間預測所使用的參考畫面不同，所以它們在解碼時需準備的資料和解碼流程也不完全相同；而相同時間分層的畫面它們畫面類型和 QP 都相同，所以

Corresponding author: yesyes78920xx@yahoo.com.tw
DOI:10.6159/IJSE.2014.(4-2).14

$\hat{k}_{DM}$ 的變化會相對穩定。接下來的小節將再個別說明這 6 個 DM 和 $\hat{k}_{DM}$ 的特性。



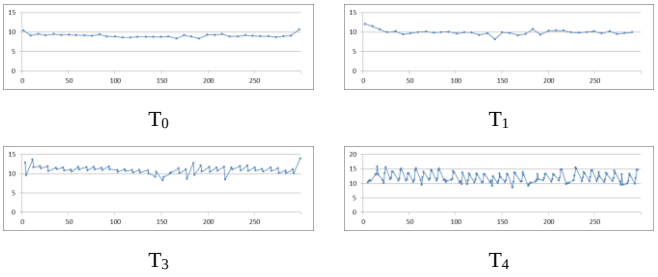
圖二：階層式 B 畫面預測架構(GOP = 8) [9]

我們使用 JM 18.3 對測試序列進行編碼，使用 Libav 10 Alpha 1 的 libavcodec 進行解碼。雖然 libavcodec 提供組合語言、硬體解碼、和多執行緒來加快解碼的速度，但是目前我們並未將上述部分納入複雜度分析，而是考慮在 PC 上的純軟體解碼。

2.2 熵解碼 (vld)

本論文探討 H.264/AVC Baseline 和 High 兩個最常用的層次(profile)，這兩個層次熵解碼 DM 使用 CAVLC (Context Adaptive Variable Length Coding)而不使用 CABAC (Context Adaptive Binary Arithmetic Coding)。我們定義 $\hat{C}_{vld}$ 為量測一張畫面在熵解碼的運算時間，n_{vld} 表示這張畫面的位元總數。在相同時間分層的畫面這個 DM 的 $\hat{k}_{vld}$ 變化很穩定，如圖三所示，所以 k_{vld} 可以用相同時間分層的前一張來預測：

$$\hat{k}_{vld} = \frac{\hat{C}_{vld}}{n_{vld}} \tag{3}$$

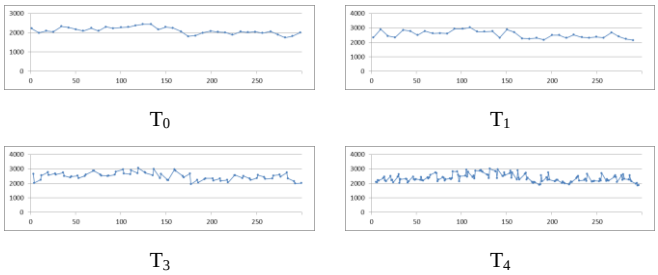


圖三：序列 Crew 量測出的 $\hat{k}_{vld}$ 在各時間分層的變化

2.3 準備附帶資訊 (sip)

這個 DM 是解碼器針對每張畫面的畫面類型和 QP，準備不同資料結構的程序，所以解碼複雜度跟畫面時間分層的相依性很高。在相同時間分層的畫面，執行一次的複雜度會很相近。令 n_{sip} 為一張畫面 MB 的總數，而每張畫面 MB 的總數通常是固定的，則 $\hat{k}_{sip}$ 變化會跟 $\hat{C}_{sip}$ 一樣(如圖四)，所以 k_{sip} 可以用相同時間分層的前一張來預測。

$$\hat{k}_{sip} = \frac{\hat{C}_{sip}}{n_{sip}} \tag{4}$$

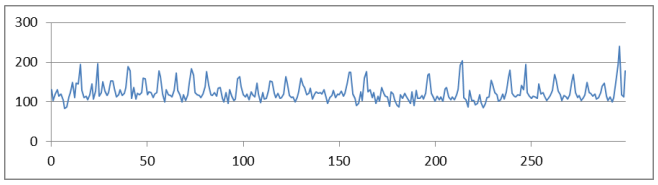


圖四：序列 Crew 量測出的 $\hat{k}_{sip}$ 在各時間分層的變化量

2.4 離散餘弦反轉換 (itrans)

此 DM 考量 4x4 區塊的反離散餘弦轉換(4x4 IDCT)。本論文仿照 JM 解碼器，利用編碼時 MB 的 CBP (Coded Block Pattern)參數推算 JM 解碼一張畫面做 4x4 IDCT 的次數 n_{itrans}。我們定義 $\hat{C}_{itrans}$ 為量測一張畫面做 4x4 IDCT 的時間，因為 libavcodec 執行 4x4 IDCT 的次數與 JM 不同，會造成 $\hat{k}_{itran}$ 偶爾有很大的值，如圖五。但由於此 DM 占整體解碼時間不多，所以我們還是將 k_{itrans} 視為一個常數 $\hat{k}_{itrans}$，定為訓練序列(training sequences)量測到的平均值(與時間分層無關)：

$$\hat{k}_{itrans} = \frac{\hat{C}_{itrans}}{n_{itrans}} \tag{5}$$

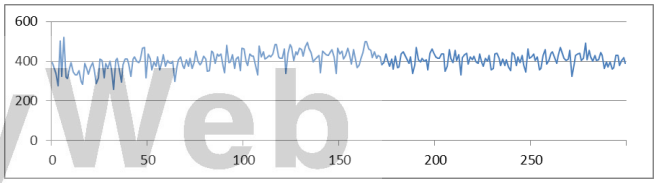


圖五：序列 Crew 量測出的 $\hat{k}_{itrans}$

2.5 畫面內預測 (intra)

雖然畫面內預測的模式(intra prediction mode)有很多種，每種模式的運算方式都有差異，但我們不特別細分它。我們定義 $\hat{C}_{intra}$ 為量測一張畫面做畫面內預測的時間，n_{intra} 為一張畫面中有幾個 MB 做畫面內預測，如圖六，$\hat{k}_{intra}$ 的變化不是很穩定，但是因為此 DM 占整體解碼時間不多，故我們將 k_{intra} 視為一個常數 $\hat{k}_{intra}$：

$$\hat{k}_{intra} = \frac{\hat{C}_{intra}}{n_{intra}} \tag{6}$$



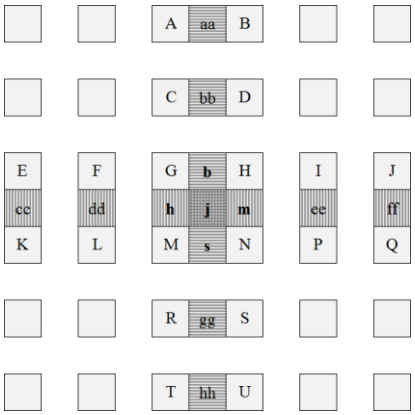
圖六：序列 Crew 量測出的 $\hat{k}_{intra}$

2.6 動作補償 (mcp)

H.264 使用 6-tap Wiener filter 內插出半像素的值，譬如在圖七中，**b** 點的計算方式如下：

$$\mathbf{b} = \text{round}\left((E - 5F + 20G + 20H - 5I + J)/32\right) \quad (7)$$

其中 E、F、G、H、I、J 為原本整數點的像素值。假如要得到 j 的值，那要先內插出 aa、bb、b、s、gg、hh 或水平的 cc、dd、…，所以進行內差的次數會因為半像素(half-pixel)位置而不同。

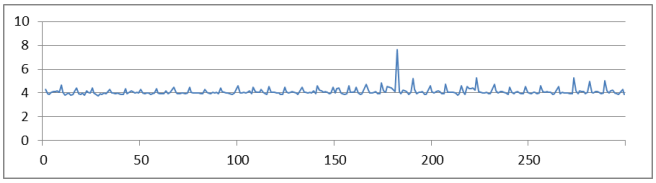


圖七：6-tap Wiener filter[1]

1/4 像素的部分 H.264 使用雙向濾波(bilinear filtering)來進行內插。畫面做動作補償時雙向濾波運算時間幾乎可以忽略不計，所以只要知道每張畫面要內插出的像素會執行幾次 6-tap Wiener filtering，便能估算出這個 DM 進行動作補償的複雜度。

我們定義 $\hat{C}_{\text{mcp}}$ 為量測一張畫面做內差運算的時間，n_{mcp} 表示一張畫面會執行幾次 6-tap Wiener filtering，如圖八，$\hat{k}_{\text{mcp}}$ 變化很穩定，所以可以將 k_{mcp} 視為一個與時間分層無關的常數 $\hat{k}_{\text{mcp}}$。

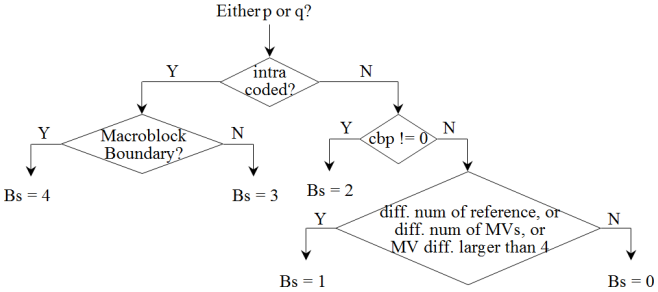
$$\hat{k}_{\text{mcp}} = \frac{\hat{C}_{\text{mcp}}}{n_{\text{mcp}}} \quad (8)$$



圖八：序列 Crew 量測出的 $\hat{k}_{\text{mcp}}$

2.7 去區塊濾波 (dblk)

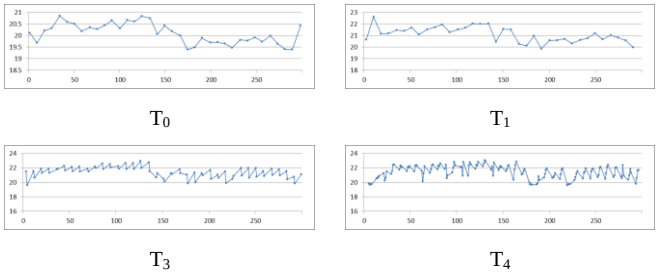
H.264 上除了畫面的邊界之外，adaptive in-loop deblocking filter 會對所有 4x4 亮度(luma) 和彩度(chroma) 區塊的邊緣進行去區塊濾波。如圖九所示，根據不同的邊界強度 (Boundary strength, Bs)，分成 3 種部不同程度的濾波，Bs = 4 為 strong filtering，Bs = 0 為 no filtering，Bs = 1、2、3 為 normal filtering。



圖九：Boundary strength decision[3]

我們沒有特別去區分 strong filtering 和 normal filtering，把他們都歸為同一類。我們定義 $\hat{C}_{\text{dblk}}$ 為一張畫面執行去區塊效應的時間，n_{dblk} 為一張畫面中的 4x4 區塊的邊緣進行去區塊濾波的次數。如圖十，在相同時間分層的畫面它們的 $\hat{k}_{\text{dblk}}$ 變化相對穩定，所以我們可以利用相同時間分層的前一張來預測 k_{dblk}。

$$\hat{k}_{\text{dblk}} = \frac{\hat{C}_{\text{dblk}}}{n_{\text{dblk}}} \quad (9)$$



圖十：序列 Crew 量測出的 $\hat{k}_{\text{dblk}}$ 在各時間分層的變化量

2.8 畫面整體複雜度

上述的 6 個 DM 在解碼一張畫面時的時間分別為 $\hat{C}_{\text{vld}}$、$\hat{C}_{\text{sip}}$、$\hat{C}_{\text{itrans}}$、$\hat{C}_{\text{mcp}}$、$\hat{C}_{\text{intra}}$、$\hat{C}_{\text{dblk}}$，把這 6 個 $\hat{C}_{\text{DM}}$ 加總起來即是一張畫面的複雜度 $\hat{C}_{\text{frame}}$

$$\hat{C}_{\text{frame}} \approx \sum_{\text{DM}} C_{\text{DM}} \quad (10)$$

C_{DM} 越接近原本 DM 實際量測的時間 $\hat{C}_{\text{DM}}$，那預測出的畫面複雜度就越準確，而 C_{DM} 可以依編碼方提供的 n_{DM} 和我們預測的 k_{DM} 估算出來，其中 k_{itrans}、k_{intra}、k_{mcp} 我們依平台定成常數，k_{vld}、k_{sip}、k_{dblk} 則是利用相同時間分層的前一張的值來預測。

$$C_{\text{DM}} = k_{\text{DM}} \times n_{\text{DM}} \quad (11)$$

三、 實驗結果

我們定義各解碼模組 DM 複雜度估計的誤差百分比：

$$\varepsilon_{DM} = \frac{\left|C_{DM} - \hat{C}_{DM}\right|}{\hat{C}_{DM}} \cdot 100\% \qquad (12)$$

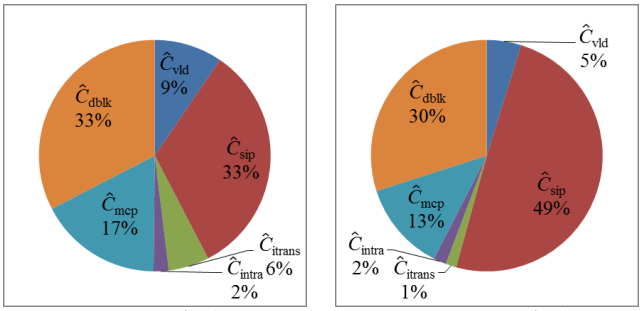
並且計算對於各測試序列估計誤差百分比的平均值和標準差，如表 I 和表 II 所示。每張畫面的誤差百分比的平均值和標準差顯示於表 I 和表 II 的最後一欄，分別只有 3.4%和 4.3%。另外，各解碼模組 DM 占整體解碼時間的百分比如圖十一所示。預估最不準確的兩個 DM 分別是離散餘弦反轉換(itrans)和畫面內預測(intra)，但是由於它們只占整體解碼時間極少的比例，所以並不會嚴重影響整體預測的準確度。

表 I
DM 複雜度估計誤差百分比(%)的平均值

Seq.	vld	sip	itrans	intra	mcp	dblk	frame
Gangster	8.4	4.3	39.8	16.3	2.4	2.0	2.5
Grass	10.2	10.2	59.3	15.0	3.7	13.1	4.0
Highway	12.5	8.3	19.5	16.7	3.9	3.1	4.0
Pourwater	13.8	5.7	34.1	10.9	3.6	3.8	3.8
Sailormen	9.7	5.3	23.6	14.7	3.2	1.8	2.5
avg.	**11.0**	**6.8**	**35.3**	**14.7**	**3.4**	**4.8**	**3.4**

表 II
DM 複雜度估計誤差百分比(%)的標準差

Seq.	vld	sip	itrans	intra	mcp	dblk	frame
Gangster	7.2	7.3	21.8	12.4	1.9	2.9	4.4
Grass	8.8	9.9	28.6	21.7	3.1	13.0	4.3
Highway	12.9	7.2	19.4	14.2	4.9	2.6	3.3
Pourwater	14.7	9.8	32.8	15.2	3.0	6.2	7.3
Sailormen	9.2	4.4	16.9	16.9	2.8	1.6	2.1
avg.	**10.6**	**7.7**	**23.9**	**16.1**	**3.1**	**5.3**	**4.3**



(a) Crew 序列 **(b) Gangster 序列**

圖十一：各 DM 占的解碼時間

本論文也與文獻[8]的方法進行比較，我們測試 CIF 解析度的測試序列(QP = 28)，其結果如表 III，由此可見[8]的方法只用了畫面解析度、每秒畫面張數、QP 3 個參數，並不能區分出不同畫面類型在解碼時的複雜度，本論文的方法區分出各個 DM 個別計算運行次數，因此對於不同的畫面類型能得到更精準的複雜度預測。

表 III
跟[8]比較各序列複雜度預測的誤差百分比

Seq.		[8]的結果	本論文的結果
Baseline profile	city	6.8%	3.4%
	crew	5.1%	3.0%
	harbor	6.8%	2.7%
	soccer	5.0%	3.0%
High profile	city	6.5%	3.0%
	crew	6.7%	3.1%
	harbor	5.5%	4.1%
	soccer	5.3%	3.2%

四、 結論

我們將 libavcodec 之 H.264 解碼器區分成 6 個解碼模組 DM，在編碼時去計算每個 DM 在解碼時執行的次數 n_{DM} ，並參照不同平台(Intel 或 ARM 的處理器)在 DM 的複雜度 k_{DM} 來預測畫面解碼的複雜度。實驗結果顯示，這些編碼參數對於複雜度的預估具有一定的可靠性；雖然有少數解碼模組其預估的準確度較差，但是由於它們只占整體解碼時間極少的比例，所以並不影響畫面整體複雜度預測的準確性。而相較於過去的文獻，本論文提出的做法誤差更小。

未來我們計劃要進一步在嵌入式平台上進行減少電量的消耗實驗，並依據預測的複雜度進行 DVFS，達到讓影片播放順暢與節省電量消耗的目的。

參考文獻

[1] I. E. Richardson, *The H.264 Advanced Video Compression Standard*, 2nd Edition, Wiley, Aug. 2010.
[2] About Libav, http://libav.org/about.html.
[3] Z. Ma, H. Hu, and Y. Wang, "On complexity modeling of H.264/AVC video decoding and its application for energy efficient decoding," *IEEE Transactions on Multimedia*, vol. 13, no. 6, pp. 1240-1255, Dec. 2011.
[4] M. Horowitz, A. Joch, F. Kossentini, A. Hallapuro, "H.264/AVC baseline profile decoder complexity analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 704-716, Jul. 2003.
[5] Z. Ma, Z. Zhang, and Y. Wang, "Complexity modeling of H.264 entropy decoding," *IEEE International Conference on Image Processing (ICIP)* , pp. 2784-2787, 2008.
[6] M. Semsarzadeh, M. J. Langroodi, M. R. Hashemi, and S. Shirmohammadi, "Complexity modeling of the motion compensation process of the H.264/AVC video coding standard," *IEEE International Conference on Multimedia and Expo (ICME)*, pp. 925-930, 2012.
[7] N. Kontorinis, Y. Andreopoulos, and M. van der Schaar, "Statistical framework for video decoding complexity modeling and prediction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 7, pp. 1000-1013, Jul. 2009.
[8] X. Li, Zhan Ma, and F. C. A. Fernandes, "Unified complexity model for H.264/AVC video processing on mobile platform," *IEEE International Conference on Image Processing (ICIP)*, pp. 2937-2940, 2012.
[9] H. Schwarz, D. Marpe, T. Wiegand, "Overview of the scalable video coding extension of the H.264/AVC standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1103-1120, Sept. 2007.