A Case of Performance Analysis of Hierarchical Parallelism on Multicore Clusters

Abelardo Rodríguez León
Instituto Tecnológico de Veracruz
Veracruz, Mexico
arleonver@yahoo.com.mx

Alberto González Téllez
Dept. de Informática de Sistemas y Computadores
Universidad Politécnica de Valencia
46022 Valencia
agt@disca.upv.es

Manuel Pérez Malumbres
Dept. de Física y Arquitectura de Computadores
Universidad Miguel Hernández
Elche
mels@umh.es

Abstract

In order to fulfill the computation requirements imposed by stressing applications, like high resolution high quality video encoding, we can apply hierarchical parallel processing techniques since actual systems support different levels of hardware parallel processing resources (clustering, multi-processor, multi-core, SIMD instruction sets, etc.). Given a delivered video quality and bitrate, the main complexity parameters are. Scalability parameters (image resolution, frame rate and latency in case of video encoding) can be pushed forward in such a way that special purpose hardware solutions may become quickly obsolete or not available. Parallel processing based on off-the-shelf components like multicore clusters is a more flexible general purpose alternative. In this work, we analyse a hierarchical parallelization of an H.264/AVC reference encoder on low cost clusters made of multicore nodes. The resulting hierarchical parallel video encoder may be configured in order to obtain a compromise between encoding throughput and latency in a specific hardware platform. We believe that our analitical procedure can be applied to other applications of similar characteristics.

1. Introduction

We are interested in cluster platforms because they are becoming a commonly available resource in an increasing number of companies and institutions that require high-performance systems able to cope with large-scale applications (i.e. high-performance web server platforms). Parallel programming on clusters is also very flexible and it allows the design of parallel video encoders adapted to almost any requirement.

Resources available on clusters vary from single to multiple CPU per node, and in every node we can have multimedia extensions in the CPUs and powerful graphic coprocessors. To make efficient use of all these computation resources we can combine different programming approaches.

- *Message passing parallelism.* Message passing runtimes and libraries (i.e. MPI [2]) allow the use of a cluster to develop parallel versions of a video encoder by inter-node and intra-node processors communication.
- *Multithread parallelism.* Multithreading (i.e. OpenMP [3]) permits to use SMP cluster nodes and multicore CPUs to reduce response time of local encoder MPI processes by parallelizing sequential code bottlenecks.
- *Optimized libraries.* Sequential code can be also optimized by using additional resources like SIMD extensions and GPUs to perform complex operations. This optimization approach can be applied by hand or using optimized libraries (i.e. Intel IPP [4], AMD ACML [5], OpenGL [6], etc).

In the literature a lot of work has been done focused on using these techniques to optimize the video encoding performance. In [17][18][16][19] authors apply the message passing parallelism to well-known standard video encoders in high-performance computing platforms (multicomputers, clusters, etc.).

More recently, an increasing interest was also centered on exploiting the available underutilized hardware present in personal computers like graphic processors [27][26] and SIMD units [14][22][4][21] in order to speedup computing intensive applications at low cost. Other works have analyzed the impact of multithreading techniques in computing platforms with certain

hardware support like Intel Hyper-threading [23] and multi-core architectures [14][25].

All of these techniques can be combined hierarchically on clusters in a top-down approach, in such a way that different parallelization levels can be orthogonal. At the highest level message passing is the technique of choice and at the lowest level we can employ SIMD optimization. At intermediate levels, message passing and multithreading are competing candidates.

In this paper we analyse a hierarchical parallelization for the H.264/AVC reference encoder. We follow a top-down approach with three levels of parallelism: GOP, slice and low level optimizations (OpenMP Multithreading, manual SIMD and automatic source code optimizations). This proposal is intended to be applied on clusters of multi-core nodes, where hundreds of CPUs may be available for parallel processing. We analyze how the available computing resources can be organized in order to achieve the maximum encoding throughput (speedup) with an encoding latency constraint. This is done by adjusting a machine parameter (group size) according to an algorithm parameter (number of slices).

The paper is organized as follows: First, in section 2, we present a hierarchical parallel video encoder, in which we combine the three levels of parallelism described in previous work [7][8]. In section 3 we estimate the performance of our hierarchical approach by means of analytical tools, particularly Little's law [28] and PAMELA [10]. In section 4 we present experimental performance results. Finally, in section 5, some conclusions and future work are drawn.

2. Hierarchical H.264/AVC parallel encoder

The main goal of the design is to get the maximum encoding throughput (or speedup) considering an encoding latency constraint. GOP-based parallelism gives almost perfect speedup with good scalability [8] due to the very low interaction among GOP encoding tasks. In this approach a GOP is sequentially encoded and then the time to deliver an encoded GOP, or latency, can be too long. Slice-based parallelism is intended to encode a GOP in parallel then it will reduce latency but the efficiency and scalability of the slice approach is much poorer than in the GOP approach [7]. Then computing resources dedicated to slice parallelism will improve latency but will worsen throughput. The problem to be solved is to obtain the minimum number of processor that we have to dedicate to slice parallelism, in order to comply with the latency constraint, with a minimum penalty over encoding throughput.

2.1. HPVC implementation

The HPVC is intended to run on clusters made up from multicore low cost processing nodes and interconnection networks. Parallelism can be local, using SMP and multicore technologies available inside a cluster node, or distributed over the cluster nodes.

We consider the cluster processors organized in groups. A group can be a subset of local processors, all the processor in a node or a set of nodes. At the first level every GOP is assigned to a processor group. Every processor group encodes its GOP independently of the other groups. When one GOP is completely encoded, the processor group is ready to encode the next available GOP in the video sequence.

Every processor group has a local manager, P0, which communicates with the global manager, P0' (see figure 1). The local manager asks for a new GOP to be encoded by its group when the current one is completely encoded. The global manager informs about the GOP assignment by sending a message with the assigned GOP number to the requesting local manager. The on demand GOP assignment method is quite simple and it gives good load balance.

Inside a processor group, the assigned GOP is processed decomposing its frames in slices (second level of parallelism), in such a way that every processor in the group processes one slice of every frame in the GOP. Once a processor has the next GOP number to encode, it can read and encode its corresponding slice on the frames belonging to that GOP.

Although there are other strategies for defining the slices, we have used the simplest one: slices are defined getting MBs in scan order in such a way that the number of MBs per slice is as much balanced as possible. When all the slices belonging to a frame are encoded they have to be integrated to build the encoded frame. Next, all

the encoded frames of actual GOP are put together to form the output bit stream.

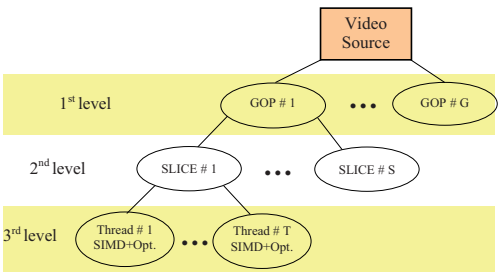

Figura 1. Hierarchical Parallelization.

Slice encoding time is optimized by code optimization performed with SIMD applied to code bottlenecks. OpenMP can also be applied if there are spare cores.

3. Performance analysis

GOP and SLICE parallel approaches can be combined in order to get the best of every one: scalability and low latency. Processing several GOPs in parallel will contribute to increase throughput. In case of live video encoding when real time response is reached, that is throughput is equal to frame rate, additional computational resources can be used to parallelize GOP encoding in order to reduce latency. This is achieved in HPVC by means of SLICE-based parallelism. To analyze the aggregated effect on performance when GOP and SLICE parallelism are combined we will use Little's law [28]: N = X*R. In order to have a precise definition of the equation's terms, we have to define what a job is. We consider a job as the encoding of one GOP. Then the equation terms are:

- *N*: Number of GOPs processed in parallel.
- *R*: Elapsed time between a GOP entering the system and the same GOP completely encoded.
- *X*: Number of GOPs encoded per second.

If we have n_P processors in the cluster and every GOP is decomposed in n_S slices, then the number of GOPs processed in parallel is $N = n_P / n_S$.

If slices are processed in parallel with efficiency E_S and if the sequential encoding time of one GOP is R_{SEQ} then GOP parallel encoding time is:

$$R = R_{SEQ} /(n_S * E_S) \quad (1)$$

Here we assume that GOP parallelization gets an efficiency close to one, as experimentally found at [8]. Finally the GOP throughput of the combined GOP-SLICE parallel encoder obtained applying Little's law is:

$$X = \frac{\frac{n_P}{n_S}}{\frac{R_{SEQ}}{n_S \cdot E_S}} = \frac{n_P}{R_{SEQ}} \cdot E_S \quad (2)$$

The effect on performance of combining GOP and SLICE parallelism is to reduce response time (latency) but throughput if affected negatively if the efficiency of slice parallelization (E_S) is significantly less than one.

As an example, let us consider a source HD video sequence at 30 fps (frames per second). We suppose that a H.264/AVC sequential encoder is able to encode one GOP (15 frames) in 30 seconds. If only one slice per frame is defined in a parallel encoder (no slice parallelism is present) then:

$$X = \frac{n_P}{R_{SEQ}} \quad (3)$$

To get real time response, X has to be equal to 30 fps or 2 GOPs/sec, then:

$$n_P = 2 \cdot 30 = 60 \text{ nodes} \quad (4)$$

GOP parallelization gives real time response in a 60-processor cluster but with 30 seconds latency. If the maximum allowed latency in the application is set to one second, then we can include slice parallelism to comply with this requirement. Let suppose E_S as 0.8. Then, How many slices do we have to define? and How many cluster processors are required?

$$n_P = \frac{X \cdot R_{SEQ}}{E_S} = \frac{2 \cdot 30}{0.8} = 75\ nodes \quad (5)$$

$$n_S = \frac{R_{SEQ}}{R \cdot E_S} = \frac{30}{1 * 0.8} = 37.5\ \text{slices} \quad (6)$$

The number of slices per frame and the number of GOPs processed in parallel have to be integers. Then, we set nS to 38 and N to 2, so the number of required processors is adjusted to 76. The estimated performance indexes are:

$$X = \frac{n_P}{R_{SEQ}} \cdot E_S = \frac{76}{30} \cdot 0.8 = 2.03\ \text{GOP/sec} \quad (7)$$

$$R = \frac{R_{SEQ}}{n_S \cdot E_S} = \frac{30}{38 * 0.8} = 0.99\,\text{sec} \quad (8)$$

From this analysis, the combined GOP-SLICE parallel encoder is able to real-time encode a 30 fps high definition video sequence on a cluster with 76 processors, observing a encoding latency less than 1 second. A key parameter is GOP encoding time R_{SEQ}, set to 30 seconds in the example. As n_S and n_P are directly proportional to R_{SEQ} any sequential code optimization will give a proportional reduction of the number of required processors and slices, improving the system cost and the encoder performance.

The other key parameter is the efficiency of the slice parallelization scheme E_S. As it is shown in the previous analysis it has as well a direct effect on n_S and n_P. In order to estimate the value of E_S we are going to apply a PAMELA model parameterized with measurements taken on a conventional cluster.

As explained in section 2, slice parallelization consists of partitioning every frame in a GOP in a fixed number of slices. Then every slice is encoded in parallel. Before proceeding with the next frame, the actual frame has to be composed and decoded to update the DPB in every processor[1]. As explained before, we implemented this synchronization by means of the MPI function Allgather [2]. Composing and decoding the encoded frame takes a noticeable amount of time (t_{CD}).

In our PAMELA model we suppose that MPI_Allgather is implemented efficiently using a binary tree, we call time taken by Allgather t_{AG}.. The number of slices processed in parallel is n_S and the mean slice encoding time is t_S. We call t_W the mean wait time due to variations in t_S and the global synchronization forced by Allgather MPI operation. The communication parameters are t_L (start up time) and t_C (transmission time of one encoded slice). Then, to encode one GOP of n_F frames by means of SLICE parallelism, we define the following PAMELA model:

```
L = seq (f=1..nF)
      par (p=1..nS)
        delay(tS); delay(tW)
      seq (i=0..log2(nS)-1)
        par (j=1.. nS)
          delay(tL + tC*2^i)
      par (p=1..ns) delay(tCD)
```

The parallel time obtained solving this model is:

$$T(L) = \sum_{i=1}^{n_f} \left(t_{S_i} + t_{W_i} + t_{AG_i} + t_{CD_i} \right) \quad (9)$$
$$t_{AG_i} = \log_2(n_s) \cdot t_L + (n_S - 1) \cdot t_{C_i}$$

So, efficiency can be computed as:

$$E_S = \frac{\frac{T_{SEQ}}{T}}{n_S} = \frac{1}{1 + \frac{\overline{t_W} + \overline{t_{AG}} + \overline{t_{CD}}}{\overline{t_S}}} \quad (10)$$

In equation (11), $\overline{t_W}$, $\overline{t_{AG}}$ and $\overline{t_{CD}}$ represent the average values along one or several GOPs. We have obtained experimental estimations of t_S, t_W and t_{CD} using JM7.6 sequential H.264/AVC encoder with the Riverbed 1280x720 video sequence at 4, 8, 16 and 32 slices (see table Measured parameters value (time in microseconds)).

Communication parameters t_L, t_C and t_{AG} have been measured running IMB benchmark [11] on a

cluster with biprocessor dual-core Opteron nodes interconnected by a Gigabit Ethernet network (Picolo cluster). Considering message sizes about the size of one encoded slice we obtain the t_{AG} values that appear on table Measured parameters value (time in microseconds). Column E_S shows the estimated values of the SLICE-based parallel encoder applying equation 11.

Tabla 1. Measured parameters value (time in microseconds).

n_S	t_S	t_W	t_{AG}	t_{CD}	E_S
4	6243	3,5	0,28	300	0.96
8	3584	16,5	1,2	325	0.92
16	1652	17,5	2,3	330	0.83
32	787	31,8	1,9	335	0.68

Considering equation 11 and the values in table Measured parameters value (time in microseconds), we notice that efficiency is not limited either by Allgather communication overhead (t_{AG}) or by synchronization wait due to differences among slice encoding time. The bottleneck is the time for composing and decoding the frame to update the DPB. This time is quite constant and then as the number of slices decreases it becomes more and more significant.
As we have shown before, another limiting factor on increasing the number of slices is the encoding performance reduction when the number of slices increases. Both factors limit the number of slices we can define and then the amount of available parallelism on the slice approach.

4. Experimental results

To evaluate the proposed H.264/AVC hierarchical parallel encoder we use two different clusters of workstations named Aldebaran and Picolo. Aldebaran is an SGI Altix 3700 cluster with 44 Itanium II nodes interconnected by a high performance proprietary network resulting in a NUMA architecture. It runs Linux RedHat 9.0 with GNU tools and MPICH (http://www.sgi.com/products/remarketed/altix3000/). Picolo is a custom-made cluster composed of 24 nodes where each node includes 2 AMD DualCore Opteron CPUs, being available a total of 96 cores. The nodes are interconnected with a Gigabit Ethernet network and a high performance Promise RAID system. Picolo runs a Rocks HPC Linux distribution. Performance measurements are obtained encoding the standard video sequence Ayersroc at SD format.

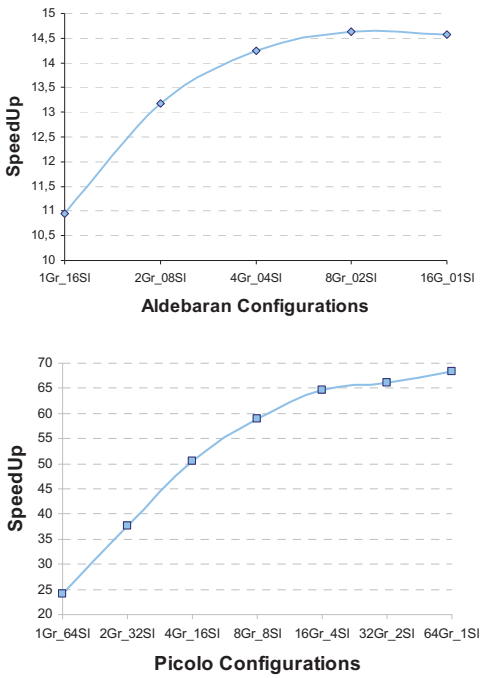

Figura 2. Speedups in Aldebaran and Picolo.

We will focus our optimization efforts on this issue and we expect a noticeable speedup improvement in the range of 8 to 32 slices.

Another effect that can be observed in figure 2, is the super speedup achieved with the cluster Picolo at certain configurations. This is due to the low level optimizations we have included only in the HPVC encoder. The sequential encoder (1Gr1Sl), used as reference for speedup computation, has not been optimized.

We have also oberved a proportional reduction of GOP encoding latency related to the node group size, as expected.

5. Conclusions and future work

After performing an analysis about the convenience of a hierarchical parallel design, we developed a three-level hierarchical parallel encoder (HPVC) based on the H.264/AVC reference source code. For each hierarchical level of parallelism, we have analyzed the parameters that define the performance behavior in terms of speedup and coding performance. Also, we have shown the potential interactions among different low level optimization techniques. Experimental results confirm the performance predictions, showing the ability to get a high-scalable and low latency H.264/AVC encoder. This is performed by adjusting the cluster configuration by means of setting up the number of processor groups (or parallel encoded GOPs) and the number of processor in a group (or number of slices in a frame).

The proposed hierarchical parallelization framework is very well suited for H.264/AVC video encoding applications, allowing the programmer to define the different hierarchical levels for a specific hardware platform and application requirements.

As future work, the identified bottleneck on the slice parallelization will be removed by further analytical and experimental work. Also, a full optimization can be performed by combining low level optimization techniques and proposing alternative source code for certain tasks of the H.264/AVC encoder that significantly reduce its computational cost.

Finally, we will try to apply the hierarchical approach to other applications with similar scalability features.

Referencias

[1] ISO/IEC 14496–10:2003, "*Coding of Audiovisual Objects—Part 10: Advanced Video Coding*," 2003, also ITU-T Recommendation H.264 "Advanced video coding for generic audiovisual services."

[2] P.S. Pacheco, "Parallel Programming with MPI", Morgan Kauf-man Publishers, Inc

[3] R. Chandra et al., "Parallel Programming in OpenMP", Morgan Kaufmann, 2000.

[4] Intel Integrated Performance Primitives, http://www.intel.com /cd/software/products/asmo-na/eng/perflib/ipp/index.htm

[5] AMD Core Math Library, http://developer.amd.com /acml.aspx

[6] OpenGL Architecture Review Board et al., "OpenGL(R) Refer-ence Manual ", 5th edition, Ed. Dave Shreiner, 2005.

[7] J.C. Fernández and M. P. Malumbres, "A Parallel implementation of H.26L video encoder", in proc. of EuroPar 2002 conf. (LNCS 2400), pp. 830, 833, Padderborn, 2002.

[8] A. Rodriguez, A. González and M.P. Malumbres, "Performance evaluation of parallel MPEG-4 video coding algorithms on clusters of workstations", IEEE Int. Conference on Parallel Computing in Electrical Engineering, pp. 354, 357, Dresden, 2004

[9] E.D. Lazowska, J. Zahorjan, G.S. Gaham, K.C. Sevcik, "Quantitative system Performance", Prentice-Hall, 1984

[10] Arjan J.C. van Gemund, "Symbolic Performance Modeling of Pa-rallel Systems", IEEE TPDS, vol 14, no 2, February 2003.

[11] Intel MPI Benchmarks: Users Guide and Methodology Descripti-on, Intel GmbH, Germany, 2004.

[12] H.264/AVC Ref. Software http://iphome.hhi.de/ suehring/tml/

[13] Intel Co. "Intel C++ Compiler for Linux Systems,User's Guide". Cap. Intel C++ Intrinsics Reference.2003

[14] Y.K. Chen, E.Q. Li, X. Zhou, S. Ge, "Implementation of H.264 encoder and decoder on personal computers", Journal of Visual Communication and Image Representation, Vol. 17, No. 2., pp. 509-532, April 2006.

[15] W. Effelsberg, R. Steinmetz. "Video Compression Techniques". Ed. Dpunkt-Verl. 1998.

[16] A. Hamosfakidis,Y. Paker, "Concurrency Analysis for Real Time MPEG4 Video Encoding",ICMCS, Vol. 2, p:862-866,1999.

[17] S. M. Akramullah, I. Ahmad and M. L. Liou, "Performance of a Software-Based MPEG-2 Video Encoder on Parallel and Distributed Systems", IEEE Transactions on Circuits and Systems for Video Technology, Vol.7, No.4, August 1997, pp. 687-695.

[18] D. Farin, N. Mache, Peter H.N. "SAMPEG, a Scene Adaptive Parallel MPEG-2 Software Encoder", SPIE Visual Communications and Image Processing, pp. 272-283, 2001.

[19] T. Olivares, F. J. Quiles, P. Cuenca, L. Orozco-Barbosa, I. Ahmad "Study of data distribution techniques for the implementation of an MPEG-2 video encoder", Parallell and Distributed Computing Systems'99. pp.537-542. MIT, Massachusetts (USA). 1999.

[20] K. Shen, L. Rowe, and E. Delp, "A Parallel Implementation of an MPEG-1 Encoder: Faster

than Real-Time," SPIE Conf. on Digital Video Compression: Algorithms and Techniques, 1995.
[21] X. Zhou, E.Q. Li, and Y.K. Chen, "Implementation of H.264 Decoder on General-Purpose Processors with Media Instructions," SPIE Conf. on Image and Video Communications and Processing, vol. 5022, pp. 224-235, Jan. 2003.
[22] V. Lappalainen, "Performance Analysis of Intel MMX technology for an H.263 Video Encoder," ACM Multimedia, pp.309-314, 1998.
[23] E.Q. Li and Y.K. Chen, "Implementation of H.264 Encoder on General-Purpose Processors with Hyper-Threading Technology," SPIE Conf. on Visual Communications and Image Processing, vol. 5308, pp. 384-395, Jan. 2004.
[24] V. Iverson, J. McVeigh, B. Reese, "Real-Time H.264/AVC Codec on Intel Architectures," Int'l Conf. on Image Processing, pp. 1541-1544, Oct. 2004.
[25] M. Roitzsch, "Slice-balancing H.264 video encoding for improved scalability of multicore decoding", ACM&IEEE international conference on Embedded software, pp. 269-278, 2007.
[26] J.Y. Hong, M.D. Wang, "High speed processing of biomedical images using programmable GPU", International Conference on Image Processing, pp. 2455-2458, Oct. 2004.
[27] F.D. Igual, R. Mayo, E.S. Quintana-Orti, "Attaining High Performance in General-Purpose Computations on Current Graphics Processors", VECPAR, pp. 406-419, 2008.
[28] J. Little, "A proof of the queueing formula L=λW", Operations Research 9:383-387, 1961.
[29] Intel Co. "Intel C++ Compiler Optimizing Applications", 2006.Lamport, L. LaTeX, a document preparation system. Addison-Wesley, 1994.