Comparative Study of Heterogeneous Multicore Scheduling Algorithms on Media Codecs

Nagendra Kumar Jamadagni[1], Aniruddh M[2],
Dr. Govinda Raju M[3] and Dr. Usha Rani K. R[4]

[1]*Student, Dept of Electronics and Communication, RV College of Engineering, India*
[2]*Student, Dept of Electronics and Communication, RV College of Engineering, India*
[3]*Assistant Professor, Dept of Electronics and Communication, RV College of Engineering, India*
[4]*Associate Professor, Dept of Electronics and Communication, RV College of Engineering, India*
[1]*nagendrajamadagni@gmail.com,* [2]*animaruthesh@gmail.com,*
[3]*govindarajum@rvce.edu.in,* [4]*usharani@rvce.edu.in*

Abstract: *All modern-day computers and smartphones come with multi core CPUs. The multicore architecture is generally heterogeneous in nature to maximize computational throughput. These multicore systems exploit thread level parallelism to deliver higher performance, but they are limited by the requirement of good scheduling algorithms that maximize CPU utility and minimize wasted and idle cycles. With the rise in streaming services and multimedia capabilities of smartphones, it is necessary to have efficient heterogeneous cores which are capable of performing multimedia processing at a fast pace. It is also needed that they utilize efficient scheduling algorithms to achieve this task. This paper compares some heterogeneous multi core scheduling algorithms available and determines which is the most optimal scheduling algorithm given various codecs.*
Keywords: Typed DAG Scheduling, Media Codecs, Heterogeneous Multicore Architecture

1. INTRODUCTION

Smartphone usage has increased exponentially in recent years and the general trend is that it is expected to continue to increase in the coming years. Along with the increasing usage, the multimedia capabilities of smartphones is also improving with each new generation of smartphones. In this regard, smartphone chipset designers are introducing heterogeneous cores like Digital Signal Processors (DSPs) and Graphical Processing Units (GPUs) so as to meet the specialized computing requirements of multimedia requirements.

These multi core CPUs are in need of good scheduling algorithms that can efficiently schedule the tasks such that the utilization of CPUs is maximized and there are fewer wasted cycles. In this regard, various different heterogeneous scheduling algorithms that are found in literature are studied and a comparison is made between them to determine which is the best available algorithm at our disposal.

Different media codecs are considered and for each codec, the encoding and decoding steps are formulated as task graphs and the worst-case response time (WCRT) for the tasks are determined with the help of the scheduling algorithm and the results are tabulated. The theoretical and practical WCRTs are shown and finally, the best overall scheduling algorithm is chosen from the given set.

2. LITERATURE SURVEY

In J.M Jaffe [1], the authors have proposed one of the first heterogeneous multicore scheduling algorithms suggested. The tasks are modeled as typed Directed Acyclic Graphs (DAGs), meaning each task is broken into a set of steps and these steps are modeled as the nodes of a DAG. The nodes are given different colours depending on which type of core the task is expected to execute on. The direction of the edges represents the precedence relations between the different tasks. This scheduling algorithm is extremely pessimistic when it comes to calculating the WCRT as it calculates the WCRT by assuming that the entire task graph is trying to execute at any given time and does not exclude the nodes that have already been scheduled and executed.

In Han et al [2], the authors have improved upon the WCRT bounds set by [1]. Instead of considering the entire task graph at each stage, the entire task graph is broken into paths from the source to the sink and the response time for each path is calculated. Then, the maximum response time so obtained is considered to be the total response time of the entire graph. This reduces the pessimistic bounds set by [1] but does not consider the fact that several nodes are common to many of the paths. And once these nodes are considered in the earlier paths, they need not be considered once again in the later paths. This oversight allows more room for improvement in the response time bounds set.

Finally, in Chang et al [3], the authors have taken into account the arguments made regarding [2] and improved the scheduling bounds for heterogeneous multi core systems. It is our wish now to use these three.

For the study of the various multimedia codecs, the textbook by Fred Halsall [4] provided the necessary insights to better understand the codecs and helped in designing their DAG models.

3. METHODOLOGY

After reviewing the available literature, we decided to model some commonly used codec algorithms as task graphs and implement the scheduling algorithm provided in [3] in the Python programming language. To test the results of our implementation, we are using the H.263 codec, JPEG codec and MP3 codec. After the modeling of the task graphs, the scheduler was used to measure the worst-case response time and the results were tabulated for comparison.

3.1. Modeling the codecs as Directed Acyclic Graphs

H.263:

The first codec which we are interested in modeling is the H.263 video compression codec. It was originally meant for low bit rate applications such as Public Switched Telephone Networks and was developed in 1996 [10]. It uses the following steps shown in Figure 1. to convert a video signal into a digitized video file. The corresponding DAG for the above codec was developed by considering all filtering, vector estimation and vector prediction operations to run on the specialized GPU core and the remaining encoding, file I/O was considered to run on the general-purpose CPU. The weights of the tasks were assigned keeping in mind the complexity of each task relative to the others. This DAG is shown in Figure 2.

MP3:

The MP3 codec is an audio compression codec based on the principle of perceptual encoding. It is a lossy compression scheme designed in the 1980s for the purposes of voice encoding and audio compression [10]. The block diagram of a typical coder and decoder is shown in Figure 3. The DAG generated from the above process is shown in Figure 4. Here, the blocks relating to filtering, and multiply and accumulate operations (MAC) are expected to run on a DSP chip. Any operations relating to file I/O and other general encoding

runs on the general-purpose CPU. The weights were once again decided based on the complexity of each task in relation to the others.

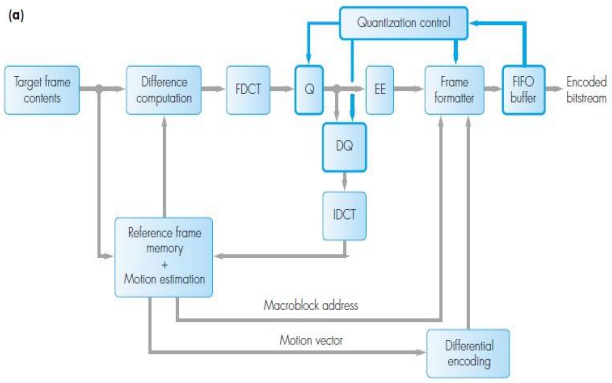

Figure 1. The block diagram of a general H.263 encoder [4]

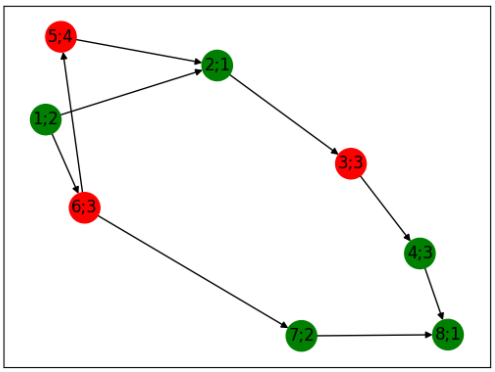

Figure 2. Task Graph for H.263 Encoder. Node 1 represents File I/O, 2 is for delta computation, 3 is DCT calculation, 4 is Quantizer, 5 is Motion estimation and prediction, 6 is Encoding, 7 is Frame Formatter and 8 is Output Write Operation.

JPEG:

The JPEG compression scheme is a lossy compression algorithm used to encode image data. It was developed by the JPEG team during the 1970s and 1980s [11]. The different steps involved in the compression and decompression using JPEG scheme is shown in Figure 5. In designing the DAG model for this, any operations relating to discrete cosine transform computation and filtering was assumed to be done on a specialized co-processor whereas general steps like entropy coding and file I/O were expected to run on a general-purpose CPU. This DAG thus designed is shown in Figure 6. The weights of the nodes were once again determined in relation to the other nodes in the Graph.

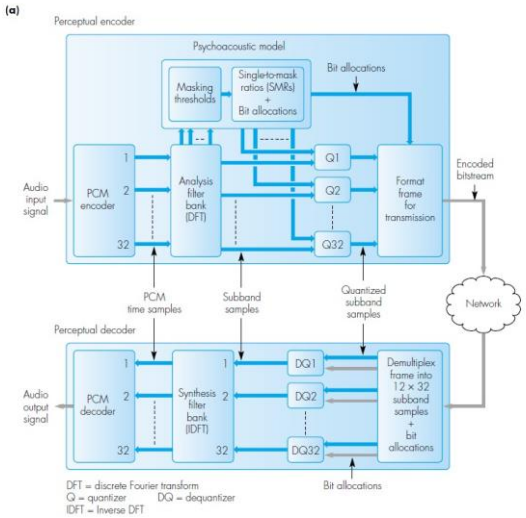

Figure 3. The block diagram of a general MP3 encoder [4]

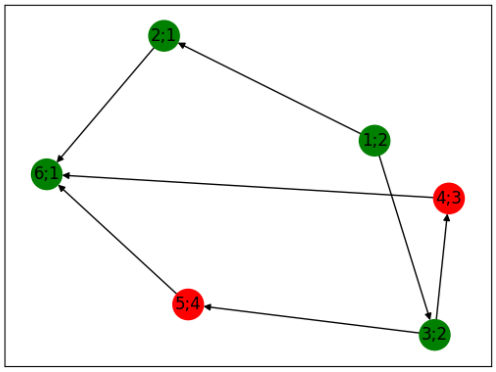

Figure 4. Task Graph for MP3 encoder. Node 1 represents File I/O, 2 is for timing control, 3 is ADC conversion, 4 is Subtraction for coding, 5 is MAC operation, 6 is Parallel to Serial Conversion of final output.

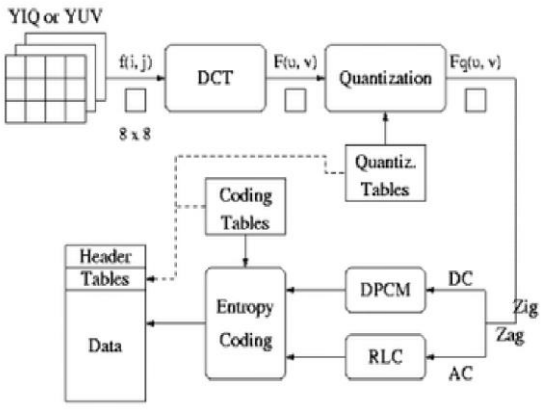

Figure 5. The schematic of a general JPEG encoder [5]

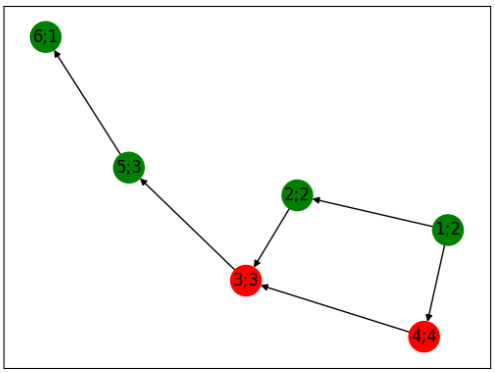

Figure 6. Task Graph for JPEG Encoder. Node 1 represents File I/O, 2 is for block preparation, 3 is Quantizer, 4 is DCT Calculation, 5 is Encoder, 6 is Output Write Operation

3.2. Scheduler Working

Once the task graphs are created, it is necessary to run the scheduler on the generated task graphs to calculate the worst-case response time for each task. The scheduler consists of 4 algorithms executed one after another to get the final result. The Figure 7 shows the first algorithm which is the unit graph transformation algorithm. It converts the nodes with weights larger than one in the graph to produce multiple nodes each of weight 1. Once the unit graph is created, we can perform the criticality allocation strategy algorithm.

Algorithm 1: Unit Graph Transformation Algorithm

Result: A unit transformed graph H which is generated from graph G

H = empty graph;
$successorList$ = empty list;
for *each vertex* v_i *in graph* G **do**
  if $v_i \in H$ **then**
    continue;
  end
  if $v_i == v_{src}$ **then**
    $H = H \cup v_{src}$;
    $successorList = successorList \cup successor(v_{src})$;
  end
  if $v_i == v_{snk}$ **then**
    $H = H \cup addEdge(v_{snk}, predecessor(v_{snk}, G)_{max})$;
  end
  while $len(successorList) > 0$ **do**
    $curNode = remove(successorList[0])$;
    for $count \leftarrow 0$, $count$ **to** $curNode_{weight}$ **do**
      $H = H \cup curNode_{count}$;
      if $count == 0$ **then**
        $H = H \cup addEdge(curNode_0, predecessor(curNode, G))$;
      else
        $H = H \cup addEdge(curNode_{count}, curNode_{count-1})$;
      end
      $successorList = successorList \cup successor(curNode)$;
    end
  end
end

Figure 7. Unit Graph Transformation Algorithm

Figure 8 shows the algorithm for criticality allocation strategy, the main idea is that it traverses each path once and identifies the nodes which are yet to be allocated a criticality set. Once the criticality set range has been allocated, the next algorithm shown in Figure 9 allocates them into the optimal criticality set. Finally, Figure 10 shows the scheduler algorithm that allocates the order of execution of different tasks.

Algorithm 2: Criticality Allocation Strategy for typed DAG

Compute $pathLength(L)$;
$L_{order} \leftarrow pathLengthOrder(L)$;
for *each unit vertex* $v_i \in l_{max}/\{v_{src}, v_{snk}\}$ do
  for *each unit node* $v_{a,b}$, $b \leftarrow 0$ to $c(v_a)$ do
    $P(v_{a,b}) = \lambda(v_{a,b}, l_{max})$;
    $P_{P_{v_{a,b}}} = \{v_{a,b}\}$;
  end
end
$H = l_{max}$;
for *each* $l \in L_{order}$ *from second to last* do
  $M = l \cap H$;
  if $M \cap l == l$ then
    continue;
  else
    for *each* $v_{i,j} of v_i \in M$ do
      Compute $\lambda(v_{i,j}, l)$ and $\lambda(v_{i,j}, M)$;
      for $v_{p,q}$ *of* $v_p \in M$ *and* $v_{i,j} \neq v_{p,q}$ do
        if $\lambda(v_{p,q}, M) - \lambda(v_{i,j}, M) == 1$ *or* $\lambda(v_{p,q}, l) - \lambda(v_{i,j}, l) > 1$ then
          $CAF(v_{i,j}, v_{p,q}, l)$;
        else
          continue;
        end
      end
    end
  end
end

Figure 8. Critically Allocation Strategy for typed DAG

Algorithm 3: Criticality Assignment Function

$N_{p(u,v)} = A(v) - B(u) - 1$;
$N_{v(u,v)} = \lambda(v, l) - \lambda(u, l) - 1$;
$f = N_{p(u,v)}/N_{v(u,v)}$;
$g = N_{p(u,v)}\%N_{v(u,v)}$;
for *each unit node* $w \in l$ ***from*** $\lambda(u, l) + 1$ **to** $\lambda(v, l) - 1$ **do**
  $A(w) = B(pre(w)) + 1$;
  $B(w) = A(w) + f - 1$;
  if $\lambda(w, l) \geq \lambda(v, l) - g$ **then**
    $B(w) = A(w) + f$;
  end
  $MIN = 0$;
  $P(w) = A(w)$;
  for *each criticality* $k \in [A(w), B(w)]$ **do**
    $P_k^* = P_k \cup \{w\}$;
    for *each criticality set* $P' \in P_k, P_k^*$ **do**
      $MAX(P') = 0$;
      for *each type* $s \in \Delta(P')$ **do**
        Compute $inf(P', s$ by using Equation 3.1;
        if $MAX(P' < inf(P', s)$ **then**
          $MAX(P' = inf(P', s)$
        end
      end
      $R(P') = 1 + MAX(P')$;
    end
    $R' = R(P_k^*) - R(P_k)$;
    if $MIN > R'$ **then**
      $MIN = R'$;
      $P(w) = k$;
    end
  end
  $P_k = P_k \cup \{w\}$;
  $H = H \cup \{w\}$;
end

Figure 9. Criticality Assignment Function

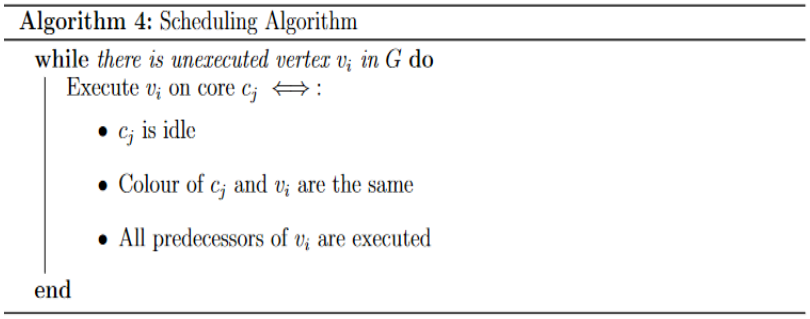

Figure 10. Scheduling Algorithm

3.3. Calculation of Worst-Case Response Time

The calculation of the worst-case response time for Jeffery's algorithm [1] is given by Equation 1.

$$R(G) \leq len(G) + \Sigma_{s \in S} (vol_s(G)/m_s) - len(G)/max_{s \in S}\{m_s\} \quad (1)$$

Where vols(G) is the total workload of the vertices of types in graph G. ms is the number of cores of types. len(G) is the longest path in graph G. The second and third terms in the equation refer to the total blocking time offered by the tasks.

Han's worst case response time is given in [2] as follows in Equation 2.

$$R(l) = len(l) + \Sigma_{s \in S} (vol_k(G)/m_s) - \Sigma_{vi \in l} c(v_i)/m_r(vi) \quad (2)$$

Where m(vi) means the number of cores on which the vertex of type (vi) can run. The first term refers to the length of path l and the remaining terms refer to the blocking time in path l. The final worst case time is the maximum of WCRTs for all paths.

Finally, Chang's worst case response time [3] is theoretically calculated based on the formula given in Equation 3.

$$R(G) \leq c(v_{src}) + c(v_{snk}) + \Sigma^{n}_{k=1} (1 + max_{s \in \Delta(Pk)}(inf(P_k, s))) \quad (3)$$

Which is the sum of the response time of each individual partition set. Here inf equation is defined in the equation in Equation 4.

$$inf(P`, s) = [(stv(P`, s) - 1)/(m_s)] \quad (4)$$

Where stv(P',s) denotes the number of blocking cores of types in the set P'.

4. RESULTS

Finally, once the task graphs were modeled, we ran the scheduling algorithms on the task graphs to get the worst-case response times. These results are tabulated in Table 1. All results assumed that there were two generalized CPU cores and 1 specialized co-processor available for task execution.

Table 1. Comparison of Scheduling Algorithms for Different Types of Codecs

Codec	Jeffery's WCRT	Han's WCRT	Chang's WCRT	Actual Time Taken
H.263	23	18	17	17
MP3 Codec	14.5	12.5	12	12
JPEG Codec	17.5	15	13	13

4.1. H.263 Codec

The output time allocation for the H.263 codec is given in Figure 11.

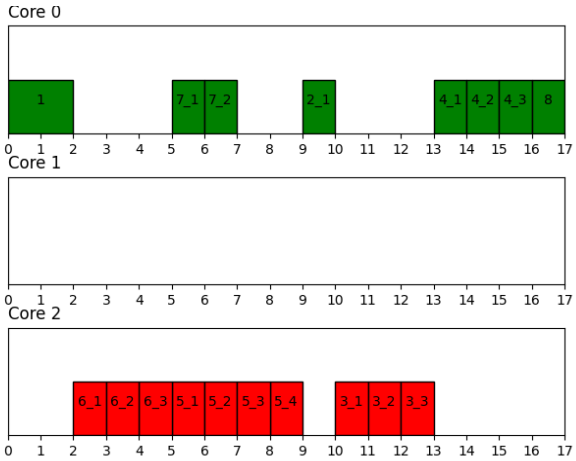

Figure 11. Time slice allocation for H.263 execution with 2 generalised CPU cores and 1 GPU core.

4.2. MP3 Codec

The output time allocation for the MP3 codec is given in Figure 12.

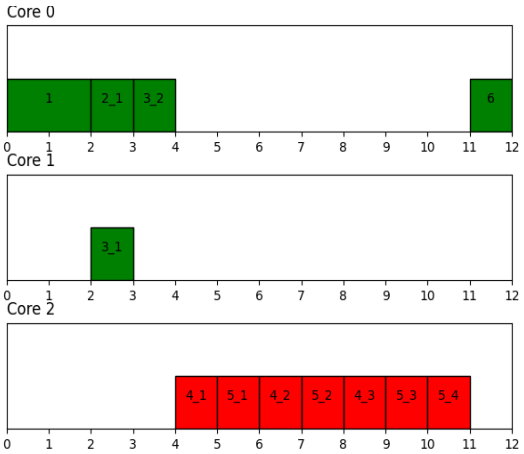

Figure 12. Time allocation for MP3 codec execution with 2 generalized CPU cores and 1 GPU core.

4.3 JPEG Codec

The output time allocation for the JPEG codec is given in Figure 13.

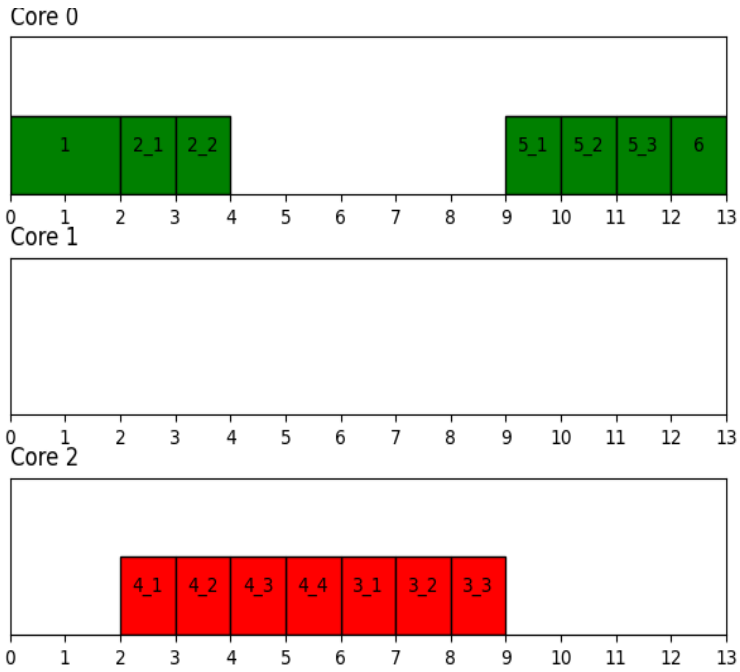

Figure 13. Time allocation for JPEG codec execution with 2 generalized CPU cores and 1 GPU core.

The inference that we can draw from the tabulated results is that Chang's algorithm is much faster than Jeffery's and Han's algorithms and hence is the optimal scheduling algorithm. The reason for this high degree of optimality is that Chang's algorithm iteratively removes nodes from the blocking paths. This significantly reduces the worst-case response time in case of task scheduling and execution.

Also, by looking at the table, it is easy to conclude that Chang's scheduling algorithm with our fast and optimized implementation in python is the best algorithm to reduce the WCRT of media codecs and hence is highly suitable for mobile system application.

5. CONCLUSION

In conclusion, we demonstrate that the partition scheduling algorithm scheme developed by Chang [3] is the most suitable algorithm for heterogeneous multicore mobile systems for scheduling multimedia tasks such as coding and decoding. With the expected rise in smartphone usage and with increased multimedia capabilities being added every year there is a need for fast paced and optimized schedulers and our implementation is a step in that direction to solve the problem.

The future scope for researchers to build off this paper is for them to design more optimal algorithms that can improve on the worst-case response times provided in this paper. There is also a possibility to model the DAG tasks in a manner that they see better fit and obtain their own results. They are also encouraged to check for other multimedia codecs available for commercial use.

6. REFERENCES

[1] J. M. Jaffe, “Bounds on the scheduling of typed task systems,”SIAM Journal onComputing, vol. 9, no. 3, pp. 541–551, Aug. 1980.doi:10.1137/0209040.

[2] M. Han, N. Guan, J. Sun, Q. He, Q. Deng, and W. Liu, “Response time bounds for typed dag parallel tasks on heterogeneous multi-cores,”IEEE Transactions on Parallel and Distributed Systems, vol. 30, no. 11, pp. 2567–2581, 2019.doi:10.1109/TPDS.2019.2916696.

[3] Shuangshuang Chang, Xufeng Zhao, Zhenyu Liu, Qingxu Deng, Real-Time scheduling and analysis of parallel tasks on heterogeneous multi-cores, Journal of Systems Architecture, Volume 105, 2020

[4] Book (Halsall2001) Halsall, F. Multimedia Communication 2001

[5] Haskell, Barry & Howard, Paul & Lecun, Yann & Puri, Atul & Ostermann, Jöern & Civanlar, M. & Rabiner, Lawrence & Bottou, Leon & Haffner, Patrick. (1998). Image and video coding - Emerging standards and beyond. Circuits and Systems for Video Technology, IEEE Transactions on. 8. 814 - 837. 10.1109/76.735379.

[6] R. Diestel, Graph Theory, ser. Graduate Texts in Mathematics. Springer Berlin Heidelberg, 2010, isbn: 9783642142789. [Online]. Available: https://books.google.co.in/books?id=KGYTSwAACAAJ.

[7] A. Silberschatz, P. Galvin, and G. Gagne, Operating System Concepts, ser. Windows XP update. Wiley, 2005, isbn: 9780471694663.[Online].Available:https://books.google.co.in/books?id=FH8fAQAAIAAJ.

[8] R. Pathan, P. Voudouris, and P. Stenstrom, \Scheduling parallel real-time recurrent tasks on multicore platforms," IEEE Transactions on Parallel and Distributed Systems, vol. 29, no. 4, pp. 915{928, 2018. doi: 10.1109/TPDS.2017.2777449

[9] T. Yang, Q. Deng, and L. Sun, \Building real-time parallel task systems on multicores: A hierarchical scheduling approach," Journal of Systems Architecture, vol. 92, pp. 1{11, 2019, issn: 1383-7621. doi: https://doi.org/10.1016/j.sysarc. 2018.10.006. [Online]. Available:https://www.sciencedirect.com/science/article/pii/S1383762118303369.

[10] Yang et al,Partition scheduling on heterogeneous multicore processors for multi-dimensional loops applications," International Journal of Parallel Programming, 2017

[11] Reference (ITU) ITU (Ed.) SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMSInfrastructure of audiovisual services – Coding of moving video

[12] Reference (ITU) ITU (Ed.) Study Group 16: AUDIOVISUAL AND MULTIMEDIA SYSTEMS JPEG Standard