Performance-Aware Decoding Algorithms For H.264 Codec On A Multi-Core Platform

Acknowledgement

This thesis is the result of research performed under the guidance of **Dr. Bhargab B. Bhattacharya** and **Dr. Ansuman Banerjee** at the Department of Advanced Computing and Microelectronics Unit(ACMU) of the Indian Statistical Institute(ISI), Kolkata. I am deeply grateful to my research advisors, **Dr. Susmita Sur-Kolay** of ACMU, ISI and **Mr. Bhaskar Karmakar** and **Mr. Prasenjit Basu** of **Texas Instruments, India** for having given me the opportunity of working as part of their research group and the huge amount of time and effort they spent guiding me through several difficulties on the way. Without the help, encouragement and patient support I received from my guides, this thesis would never have materialised.

Nibedita Moitra

July 15, 2011

Advanced Computing and Microelectronics Unit

Indian Statistical Institute

203 B T Road, Kolkata

Pin 700 108, India

Certificate

This is to certify that the thesis titled **Performance-Aware Decoding Algorithms For H.264 Codec On A Multi-Core Platform** submitted by **Nibedita Moitra** to the Advanced Computing and Microelectronics Unit in partial fulfilment for the award of the degree of Master of Technology is a bona fide record of work carried out by him under our supervision and guidance. The thesis has fulfilled all the requirements as per the regulations of this Institute and, in our opinion, has reached the standard needed for submission. The results embodied in this thesis have not been submitted to any other university for the award of any degree or diploma.

Dr. Bhargab B. Bhattacharya
Advanced Computing and Microelectronics Unit
Indian Statistical Institute
Kolkata

Dr. Ansuman Banerjee
Advanced Computing and Microelectronics Unit
Indian Statistical Institute
Kolkata

Keywords: H.264 Decoder, Macroblock, Parallelization, Video Decoding, Slice

Contents

List of Figures

Chapter 1

Introduction

Contents

The latest video compression standard, $H.264$ (also known as MPEG-4 Part 10/AVC for Advanced Video Coding) [3] is expected to become the video standard of choice in the coming years.

$H.264$ is an open, licensed standard that supports the most efficient video compression techniques available today. Without compromising image quality, an $H.264$ encoder can reduce the size of a digital video file by more than 80% compared to the $MotionJPEG$ format and as much as 50% more than with the $MPEG-4Part2$ standard. This means, much less network bandwidth and storage space are required for a video file. Or seen another way, much higher video quality can be achieved for a given bit rate.

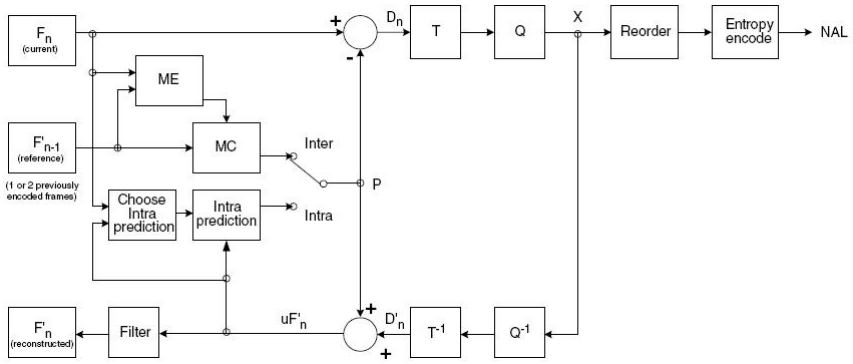
1.1 Motivation

The $H.264$ decoder and encoder both have a sequential, data dependent flow as shown in Figure 1.1. This property makes it difficult to leverage the potential performance gain that could be achieved by the use of emerging many core processors.

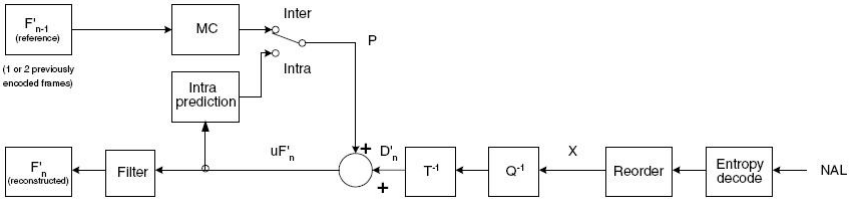
Multimedia applications remain important workloads in the future and demand high speed video coding and decoding. Their performance efficiency should increase with the increase in number of processor cores. A central question is whether $H.264$ decoder can scale to such large number of cores.

One possible way of increasing efficiency is to identify data parallelism within one or more blocks of the encoder/decoder flow. We can assign independent data to the multiple cores available in the processor to achieve speed-up in execution.

We can have dedicated Silicon implementations to achieve this. But hardware implementation is costly and we need different implementation for each new video compression standard. That is why we need a parallel software implementation of $H.264$ codec that can perform as efficiently as the hardware implementation and can run on different hardware platforms. If the hardware platform changes the software implementation needs smaller amount of change than dedicated silicon implementation.



(a) Encoder



(b) Decoder

Figure 1.1: Block diagram of H.264 encoder and decoder

1.2 Scope of the Thesis

In this project, we consider an $H.264$ decoder and explore the possibilities of parallelism in the Intra prediction block.

There are a couple of reasons behind taking up the decoder (and not the encoder) as part of this parallelization effort. Firstly, the encoding problem is a natively parallel one, and hence, lends itself more naturally to a parallelized execution environment and there are already numerous successful attempts in this direction. However, the decoding algorithm poses certain challenges

to parallelization. Secondly, there is a decoding step inside the encoder as well and therefore, any success in parallelizing the decoder would naturally expedite the encoder as well.
The slice is now the basic independent spatial element in $H.264$. This prevents an error in one slice from affecting other slices. Each block in the decoder receives as input a complete Slice. Each Slice is made up of multiple Macroblocks(MBs). If we can find data independence between the constituents of a slice i.e between MBs, we can considerably speed-up the processing of each block in the decoder. In this thesis we have performed this analysis on the Intra prediction block.

1.3 Organization of the Thesis

In Chapter 2 we have describe the Intra-prediction process used in H.264 video decoding and how this creates data dependency between Macroblocks of a video slice. In Chapter 3 we give a brief description of multi-core architectures and how the $H.264$ can fit in such an architecture.In Chapter 4 we describe in brief the survey of other related works in this area done by us.In Chapter 5 we describe our objective and the Macroblock scheduling problem in detail and derive a problem formulation. In Chapter 6 we describe the various Macroblock scheduling strategies proposed by us. In Chapter 7 we have given the performance plots of all the proposed strategies based on various parameters. Finally in Chapter 8 we have analysed the performance of all scheduling strategies and described the scenario under which each of these strategies can be best used. We have also described the scope of future work from this thesis.

Chapter 2

Overview of H.264 Intra Prediction

Contents

2.1 Slices and Macroblocks

A *H*.264 video picture can be partitioned into independent spatial units called Slices. A slice is the smallest independent spatial unit in *H*.264 codec. Each slice can be further divided into 16 pixel by 16 pixel spatial area called Macroblocks(MBs).This has been illustrated in Figure 2.1

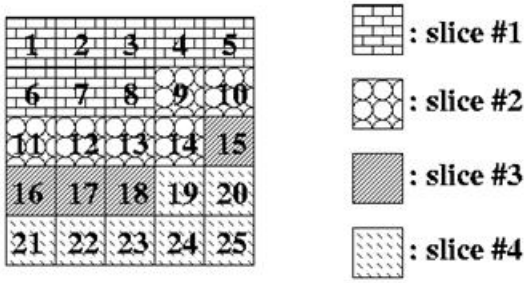


Figure 2.1: Structure of slice

2.2 Intra Macroblock Data Dependency

Every MB in a slice is dependent on its neighbouring MBs for intra-prediction as shown in Figure 2.2. CurMbAddr is the current MB and MbAddrA, MbAddrB, MbAddrC and MbAddrD are the left, top, top-right and top-left neighbours of the current MB respectively [3, sec. 6.4.8]. The number of pixel data needed from each of these neighbours is dependent on the intra prediction mode of the current MB.

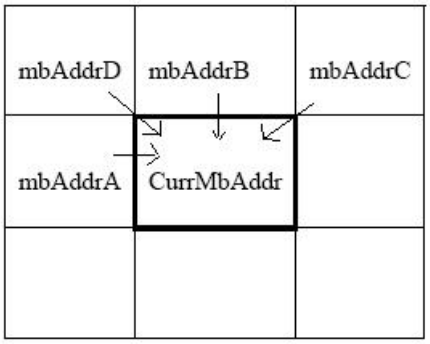


Figure 2.2: MB data dependency

2.3 Intra Prediction Modes and Data Dependency

In intra mode a prediction block P is formed based on previously encoded and reconstructed blocks and is subtracted from the current block prior to encoding.
There are three types of intra prediction modes for luma and one for chroma as follows

- For Luma
- – Intra 4x4
  – Intra 8x8
  – Intra 16x16
- For Chroma
- – Intra 8x8

2.4 Intra 4x4 Prediction Mode and Dependency Graph Formation

The current MB is divided into 16 4x4 blocks [1, sec. 6.4.6]. Each block can have one of prediction mode as specified in Figure 2.4 (a). Some bordering pixels of the neighbouring MBs might be used to predict the samples of the current MB. The number of pixels from the neighbouring MB needed for prediction of current MB sample prediction is the edge weight of the edge from the neighbouring MB to the current MB in the dependency graph of the current slice.

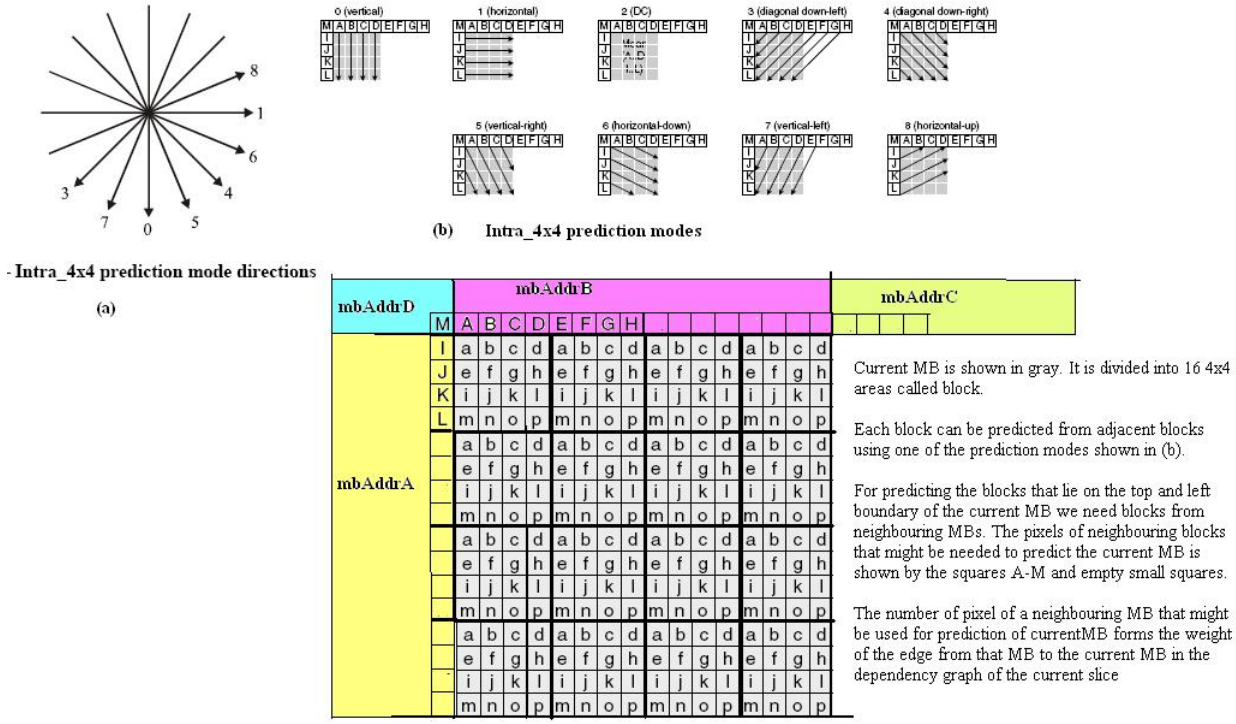


Figure 2.3: Intra 4x4 Prediction Mode and Dependency Graph Formation

2.5 Intra 8x8 Prediction Mode and dependency graph formation

The current MB is divided into 4 8x8 blocks. Each block can have one of prediction modes as specified in Figure 2.4 (a). Some bordering pixels of the neighbouring MBs might be used to predict the sample of the current MB. The number of pixels from the neighbouring MB needed for prediction of current MB sample prediction is the edge weight of the edge from the

neighbouring MB to the current MB in the dependency graph of the current slice.

2.6 Intra 16x16 Prediction Mode and Dependency Graph Formation

The various Intra 16x16 Prediction Modes are shown in Figure 2.6.The entire current MB samples are predicted from the neighbouring MB samples.

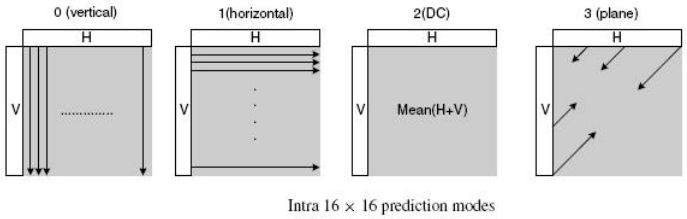


Figure 2.4: Intra 16x16 Prediction Modes

2.7 Macroblock Data Dependency Tables

The following are distinct possible ways the current MB may need samples from neighbouring MBs for its own sample prediction. Each set of four numbers separated by commas represents a unique data dependency from neighbouring MBs. The first ,second,third and fourth values represent the number of sample data needed from mbAddrA, mbAddrB, mbAddrC and mbAddrD respectively.

0,16,0,0	10,8,0,0	9,8,4,0	13,0,0,0	4,16,0,1	4,13,0,1
4,16,0,0	13,8,0,0	12,8,4,0	16,0,0,0	8,16,0,1	8,13,0,1
5,16,0,0	0,12,4,0	10,8,4,0	4,4,4,0	9,16,0,1	9,13,0,1
8,16,0,0	4,12,4,0	13,8,4,0	8,4,4,0	12,16,0,1	12,13,0,1
9,16,0,0	5,12,4,0	0,9,0,0	9,4,4,0	13,16,0,1	13,13,0,1
12,16,0,0	8,12,4,0	4,9,0,0	12,4,4,0	16,16,0,1	16,13,0,1
10,16,0,0	9,12,4,0	5,9,0,0	13,4,4,0	4,12,0,1	4,4,0,1
13,16,0,0	12,12,4,0	8,9,0,0	16,4,4,0	8,12,0,1	8,4,0,1
0,12,0,0	10,12,4,0	9,9,0,0	4,5,0,0	9,12,0,1	9,4,0,1
4,12,0,0	13,12,4,0	12,9,0,0	8,5,0,0	12,12,0,1	12,4,0,1
5,12,0,0	0,13,0,0	10,9,0,0	9,5,0,0	13,12,0,1	13,4,0,1
8,12,0,0	4,13,0,0	13,9,0,0	12,5,0,0	16,12,0,1	16,4,0,1
9,12,0,0	5,13,0,0	0,13,4,0	13,5,0,0	4,16,4,1	4,8,4,1
12,12,0,0	8,13,0,0	4,13,4,0	16,5,0,0	8,16,4,1	8,8,4,1
10,12,0,0	9,13,0,0	5,13,4,0	4,9,4,0	9,16,4,1	9,8,4,1
13,12,0,0	12,13,0,0	8,13,4,0	8,9,4,0	12,16,4,1	12,8,4,1
0,16,4,0	10,13,0,0	9,13,4,0	9,9,4,0	13,16,4,1	13,8,4,1
4,16,4,0	13,13,0,0	12,13,4,0	12,9,4,0	16,16,4,1	16,8,4,1
5,16,4,0	0,4,0,0	10,13,4,0	13,9,4,0	4,8,0,1	4,9,0,1
8,16,4,0	4,4,0,0	13,13,4,0	16,9,4,0	8,8,0,1	8,9,0,1
9,16,4,0	5,4,0,0	16,12,0,0	16,13,0,0	9,8,0,1	9,9,0,1
12,16,4,0	8,4,0,0	16,8,0,0	16,13,4,0	12,8,0,1	12,9,0,1
10,16,4,0	9,4,0,0	16,12,4,0	4,10,0,0	13,8,0,1	13,9,0,1
13,16,4,0	12,4,0,0	16,4,0,0	8,10,0,0	16,8,0,1	16,9,0,1
0,8,0,0	10,4,0,0	16,8,4,0	9,10,0,0	4,12,4,1	4,13,4,1
4,8,0,0	13,4,0,0	16,9,0,0	12,10,0,0	8,12,4,1	8,13,4,1
5,8,0,0	0,8,4,0	4,0,0,0	13,10,0,0	9,12,4,1	9,13,4,1
8,8,0,0	4,8,4,0	8,0,0,0	16,10,0,0	12,12,4,1	12,13,4,1
9,8,0,0	5,8,4,0	9,0,0,0	16,16,0,0	13,12,4,1	13,13,4,1
12,8,0,0	8,8,4,0	12,0,0,0	16,16,4,0	16,12,4,1	16,13,4,1

Figure 2.5: Data dependency on adjacent MBs for intra 4x4 Prediction Mode

0,16,0,0	8,8,8,0
8,16,0,0	16,8,8,0
9,16,0,0	8,9,0,0
0,8,0,0	16,9,0,0
8,8,0,0	16,16,0,0
9,8,0,0	16,16,8,0
0,16,8,0	8,16,0,1
8,16,8,0	16,16,0,1
9,16,8,0	8,8,0,1
16,8,0,0	16,8,0,1
8,0,0,0	8,16,8,1
16,0,0,0	16,16,8,1

Figure 2.6: Data dependency on adjacent MBs for intra 8x8 Prediction Mode

0,16,0,0
16,0,0,0
16,16,0,0
16,16,0,1

Figure 2.7: Data dependency on adjacent MBs for intra 16x16 Prediction Mode

Chapter 3

A Parallel Software Architecture for H.264 Decoder

Contents

The objective of this work is to extend a single-threaded software implementation of a *H*.264 decoder to support multicore platforms, or platforms containing multiple processing cores on a single chip. This research is important as multicore platforms are quickly becoming ubiquitous in the desktop, server, and embedded domains including settings where real-time constraints must be satisfied.

Efficient distribution of the *H*.264 decoding algorithm among multiple processing cores is a nontrivial task. For using the available processing resources efficiently, an equally balanced distribution of the decoder onto the hardware units must be found. The system designer has to consider data dependency issues as well as inter-communication and synchronization between the processing units.

One of the most promising techniques for a parallel *H*.264 decoder is the parallelization at the level of MBs (MBs), in which small blocks of the video frame are processed in parallel. This type of parallelization has been presented in theoretical and simulation analysis as scalable and efficient. In *H*.264 usually MBs in a frame are processed in raster scan order, which means starting from the top left corner of the frame and moving to the right, row after row. To exploit parallelism between MBs inside a frame it is necessary to take into account the

dependencies between them as discussed in section 2.7. In H.264, motion vector prediction, intra prediction, and the de-blocking filter use data from neighbouring MBs defining a structured set of dependencies. MBs can be processed out of scan order provided these dependencies are satisfied. It is important to mention here that due to the sequential behaviour of the entropy decoding kernel, we have decoupled it from the rest of the macroblock decoding steps in our parallelization process.

In this work, we investigate the performance scalability of Macroblock-level parallelization for the H.264 decoder in a multi-core environment where multiple processing cores, each having a fixed amount of small local cache and having access to a larger shared cache, are placed on the same chip. Specifically, our focus is on reducing the access of shared caches and inter-core communication by increasing the level of reuse for local caches, while ensuring the dependency constraints of a given workload. In multicore platforms, reducing shared cache access can result in decreased execution times and energy consumption, which may allow a larger workload to be supported or hardware requirements (or costs) to be reduced.

To this end, our objectives are as below. Given a system of N cores, each with a fixed amount of local memory and each having access to a shared memory, our objectives are as follows:

- to develop a scheduling and allocation strategy for multiple MBs with a target of minimizing off-chip access, while satisfying dependency constraints

3.1 Multi-Core Architectures

In this work, we assume a multi-core architecture as the underlying model of execution. In multicore architectures, multiple processing cores are placed on the same chip. Most major chip manufacturers have adopted these architectures due to the thermal- and power related limitations of single-core designs. Dualcore chips are now common place, and numerous four- and eight-core options exist. Further, per-chip core counts are expected to increase substantially in the coming years. For example, Intel has claimed that it will release 80-core chips as early as 2013. Additionally, Azul, a company that creates machines for handling transaction-oriented workloads, currently offers the 54-core Vega 3 processor, which is used in systems with 864 total cores. The shift to multicore technologies is a watershed event, as it fundamentally changes the standard computing platform in many settings to be a multiprocessor.

Resource sharing in multi-cores comes in many forms. First of all, it is quite common for the processor cores to be connected by a common bus. Moreover, even though the processors have

their own private caches, current multi-core architectures often share the lowest-level cache (i.e., the level of cache that is most distant from the cores, and closest to main memory). This architecture is fairly commonthe Sun UltraSPARC T1 and T2 processors have a lowest-level L2 cache shared by eight cores, and the recentlyreleased Intel Core i7 chip contains a lowest-level L3 cache shared by four cores (in this case, each core contains private L1 and L2 caches).

3.2 The H.264 Decoder

The H.264 decoding process can be divided into two fundamentally different parts, the parser and the reconstructor. The parser includes context calculations for the context-adaptive part of the entropy decoding process, the low-level entropy decoding functions (binary-arithmetic or variable length coding) and the reconstruction of the macroblocks syntax elements such as motion vectors or residual information.

3.2.1 Parallelizing the H.264 Decoder

The entropy decoding part is highly sequential with short feedback loops at bit- and syntax element-levels. Furthermore, the conditional calculations of the entropy encoders context information require short branch executions of the executing processor and efficient data access. This makes data-parallel processing hardly usable in these decoding stages and functional splitting is preferable.

This work therefore focuses on the reconstructor module of the H.264 decoder, which provides a wide range of possibilities for exploiting data-parallelization. In the reconstruction part, the H.264 decoder uses the parsed syntax elements to generate intra or inter predictions for each macroblock. The residual coefficients are inverse transformed (IDCT) and added to the macroblocks prediction. Finally, an adaptive deblocking filter is applied to the outer and inner edges of the sub-blocks of a macroblock in order to remove blocking artifacts.

The H.264 reconstructor module can be parallelized either by task-level or data-level decomposition. In task-level decomposition the functional partitions of the algorithm are assigned to different processors. Scalability is a problem because it is limited to the number of tasks, which typically is small. In data-level decomposition the work (data) is divided into smaller parts and each part is assigned to a different processor. Each processor runs the same program but on different (multiple) data elements (SPMD). In H.264 data decomposition can be applied to different levels of the data structure. This work focuses only on MB level parallelism for intra

prediction in I slices.

Chapter 4

Survey Of Related Works

Different data level decompositions are possible on H.264 decoder [4]. Frame level parallelism(FLP) has very little scalability since there are very few B frames between P frames. In *H*.264 B frames can also be used as reference. This reduces FLP even more. Slice level parallelism depends on the size and number of slices/frame. These parameters are dependent on the encoder. Therefore load balancing and low scalability are its issues. In MB level parallelism(MLP) two strategies 2-D and 3-D wave are discussed. FLP can be combined with MLP dynamically and measure performance improvements. Study reveals 3-D wave provides huge number of parallel MBs and requires large memory bandwidth which is even beyond the scope of future many core CMPs. So the performance of 3-D wave is studied under limited resource availability conditions.

Even with limited resources 3-D wave [5] has the potential of utilizing the computational power available provided sufficient memory bandwidth to support constant number of frames in flight. The performance gain of 3-D wave is much more than 2-D wave technique. The performance of 2-D wave is highly affected by the frame resolution and scalability is not constant throughout the decoding of a frame. To overcome these drawbacks 3-D technique is exploited along with 2-D technique. It supports hundreds of frames in flight and thousands of MBs to be processed in parallel given sufficient memory bandwidth and a good frame scheduling policy.

However the scalability of 3-D decreases for large number of cores because cache thrashing occurs. The scalability also decreases at the start or end of a sequence since little parallelism is available at that time. For higher resolution video no relevant speed-up is achieved unlike 2-D. FS and FP achieve lower frame latency and higher scalability at the cost of increased memory traffic. Increase in average memory latency decreases performance gain (AML of 40-50 cycles is acceptable). L1 cache should be at least 64KB for higher number of cores. The overhead of

TLP must be limited to maximum 30% of the MB decoding time. Otherwise the performance degenerates drastically with increased number of cores.

The 3-D and 2-D implementation process [6] shows 3-D is much more scalable than 2-D. Scalability of 2-D largely depends on the resolution of video. 3-D is free from this drawback. Once the reference MB of a MB in another frame is completed the current MB may start processing. But this may lead to large number of Frames on the fly and demand large memory bandwidth. This problem is solved my introducing the concept of subscription MB which limits the number of Frames in flight. A Frame scheduling policy which introduces Frame priority helps reducing the frame latency. 3-D is implemented on an existing 2-D implementation. A feature called Tail submit is used to minimize the spatial dependency of MBs and reduce memory read/writes.

Each video frame goes through 3 stages- parse, render and filter. Chong et al. concentrates on parallelizing the first 2 stages [7]. Parsing is a major bottleneck in parallelization. To overcome this, a preparsing strategy has been applied and some of recently pre-parsed frames are buffered and later stages are performed on them in parallel. This paper compares different MB scheduling strategies greedy dynamic, static compile time and run time. Among the three greedy dynamic fails to preserve MB dependencies, static compile time fails to handle input dependent data. The run time MB scheduling preserves both.

Advantages of this technique is the runtime scheduling matches the ideal scheduling most closely. There are different data dependencies between MBs a) for inter prediction dependencies between Mbs belonging to different frames and b) between Mbs in the same frame for Intra prediction. Sun et al. introduce a MBRP algorithm [8]. It partitions a frame into partitions consisting of consecutive columns. Each partition is then assigned to a separate processor. The algorithm proceeds by wave-front method.

Each partition contains approximately equal number of consecutive non-overlapping columns. So load balancing is achieved easily. Amount of data exchanged between processors is smaller. However each partition must contain at least 2 consecutive MB columns. So highest speed-up possible is restricted by the resolution of the video source.

Kue-Hwan Sihn et. al. proposed a pipeline structure where Variable Length Decoding(VLD) is overlapped with Motion Compensation(MC) stage. VLD and MC share the same task queue. These stages behave like the producer consumer model. But the speed of VLD is a bottleneck. The 2-D wave-front technique is used both for Inter and Intra frames. Limited shared memory and off-chip memory bandwidth in multi-core processors are a bottleneck. As the number of cores increase the speed-up gradually decrease because the multiple threads that are dependent

on VLD occupy much of the shred memory. So speed-up of VLD is essential. IP and MC deal with small data partitions and they have small load imbalance. But De-blocking filters process large partitions and so they suffer from heavy load imbalance.

Software memory throttling method tries to meet the gap between VLD and MC processing speeds.De-blocking of Inter takes 3 times times as compared to that of Intra. While partitioning workload for De-blocking filter we have to consider the Mb-type as well to ensure fair load balancing.

A technique for Group Of Pictures(GOP)-level parallelization technique has been developed by Ahmet Grhanl et. al [10]. Each processor is assigned a GOP. GOPs are independent of each other and can be decoded simultaneously. The start point of each GOP is written in the video stream header by the encoder. Each processor in the decoder reads the header and chooses the part it is supposed to decode. This method gives linear speed-up in case of small GOP sizes but deteriorates as GOP size increases. For large GOP the memory requirements of the processors increases so more cache pollution occurs. So linear speed-up is not possible.

However this technique demands high memory resource and has long latency problems. One processor manages the scheduling and the others do parallel decoding. So with P processors only $P-1$ times maximum speed-up may be achieved. The scheduler may easily become insufficient for feeding the increasing number of parallel working cores.

The maximum speed-up achievable using MB level parallelism is quite high. But it assumes constant MB processing time and no synchronization overhead. When these factors are taken into account maximum speed-up possible is reduced significantly. Mauricio Alvarez Mesa et.al. compares the various scheduling approaches of MBs to cores [11]. The static scheduling is good only when MB processing time is constant(which is impossible in real cases). The dynamic scheduling exploits more parallelism but encounters huge overhead due to a centralised task queue read/write. Dynamic scheduling with tail submit is by far the best scheduling approach. Entropy decoding is entirely serial ,so it is decoupled from the parallel part of the decoder and a buffer is inserted between the modules to meet the speed differences of the two parts.

Chapter 5

Macroblock Scheduling Problem

Contents

5.1 Objective

Given a system of N processors, each with a fixed amount of local memory and each having access to a shared memory, our objectives are as follows:

- Develop a scheduling and allocation strategy for scheduling multiple MBs with a target of minimizing off-chip access, while satisfying dependency constraints.

5.2 Performance Evaluation Parameters

The effectiveness of our proposed strategies will be measured in terms of the following factors:

- Measure of multi-core utilization (speed gain over single core)
- Measure of energy efficiency(power dissipation saving over high frequency single core processor with the help of multi-core processors)
- Load balancing across all cores

- Speed-up performance consistency

5.3 Assumptions

- Incoming video stream is pre-parsed and all slices extracted and stored in a buffer.
- Processing time of a MB is the maximum of time needed for Intra prediction, Inter Prediction, inverse Scale and Transform, and loop filter.
- All MB scheduling strategies discussed are non-preemptive and static.
- Slices are rectangular (m rows $\times n$ cols)
- Processors are homogeneous
- Underlying hardware
  - Core 2 quad processors(e.g intel Q9100)
  - L1 cache
    - 4×32 KB instruction cache
    - 4×32 KB data cache
  - L2 cache
    - 2×6 MB
- Intra core communication latency
  - L1 cache latency
    - 1-2 CPU cycles
  - L2 cache latency
    - 15 CPU cycles
  - Front side bus latency
    - 3 CPU cycles
  - Hardware diagram Figure 5.1

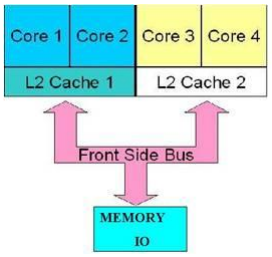


Figure 5.1: Hardware diagram

5.4 Problem Formulation

Given

set of tasks $T : T_i$ has processing time e ,which depends on the task is known a priori for each MB before scheduling starts. T_i has variable memory access time ma_i, Total execution time for T_i is e_i

$$e_i = e + ma_i, 1 <= i <= m \times n \tag{5.1}$$

set of processors $P : P_i, 1 <= i <= p$,

time axis Z ,

Dependency precedence graph $G : G = (V, E)$ where $|V| = m \times n$

if ordered pair $(v_i, v_j) \in E$,then T_j depends on $T_i, 0 <=$ indeg$(v_i) <= 4, 0 <=$ outdeg$(v_i) <= 4, \forall v_i \in V$ (5.2)

Calculate ma_i as follows. Let task T_i read d_{i1}, d_{i2}, d_{i3} pixels from Tier 1,2 and 3 memory respectively. Then

$$ma_i = d_{i1} \times ml_1 + d_{i2} \times ml_2 + d_{i3} \times ml_3 \tag{5.3}$$

Find a schedule $S : T \to P \times Z$,where $S(T_i) = (P_j, Z_r)$ means task T_i starts on processor P_j at time Z_r and define relations π and τ as follows

$$\pi : T \to P, \text{ where } \pi(T_i) = P_i, \text{ iff } S(T_i) = (P_i, t_i) \tag{5.4}$$

$$\tau : T \to Z, \text{ where } \tau(T_i) = t_i, \text{ iff } S(T_i) = (P_i, t_i) \tag{5.5}$$

such that

- **Task dependency constraint**

$\forall T_i, T_j \exists$ ordered pair $(v_i, v_j) \in E$ such that $\tau(T_i) = t_i$ and $\tau(T_j) = t_j$ then

$$t_i + e_i <= t_j \tag{5.6}$$

- **One task per processor at any instant constraint**

$\forall T_i, T_j$ such that $\pi(T_i) = P_i, \pi(T_j) = P_j, \tau(T_i) = t_i, \tau(T_j) = t_j$, then

$$P_i = P_j \tag{5.7}$$

$$[t_i, t_i + e_i] \cap [t_j, t_j + e_j] \neq \emptyset \text{ ,iff } i = j \tag{5.8}$$

Our objective is to find a schedule to minimize

$$\max\{\tau(T_j) + e_j : v_j \in V_1\} \text{ where } V_1 = \{v : v \in V, \text{ v is a sink vertex}\} \tag{5.9}$$

5.5 Tasks Performed in a Typical Decoding Cycle

- L1 read
  - First step of decoding cycle. A MB reads some data about a neighbouring MB from L1 cache.
  - Latency 2 CPU cycles
  - Among the top,top-left,top-right and left neighbours, the one who was assigned to the same core as the current core, that MB s data is available in L1
- L2 read
  - Second step of decoding cycle. A MB reads some neighbourhood data from shared L2 cache if it is not in L1.
  - Latency 15 CPU cycles
  - Among the top,top-left,top-right and left neighbours, the one who was assigned to the other core on the same die, that MB s data is available in L2
- The third step of decoding cycle is the sample prediction process of the MB excluding any communication overhead.
- Front side bus

  - Fourth step of decoding cycle. A MB writes data needed by any of its right,bottom,bottom-right, bottom-left neighbours via FSB to the L2 cache of the other die.
  - Latency 3 CPU cycles
  - If MB in core 1 or 2 has neighbours assigned to core 3 and 4 then the dependent data between them must travel through FSB.

- L2 write
  - Last step of decoding cycle. A MB writes some data needed by a neighbouring MB to shared L2 cache of the same die.
  - Latency 15 CPU cycles
  - Among the right,bottom,bottom-right,bottom-left , the one who was assigned to the other core of the same die , that MB s data is written to L2
  - Among the right,bottom,bottom-right,bottom-left , the one who was assigned to a core of the other die , that data is transmitted via FSB and written to L2 of the other die.
  - Here we assume L1 and L2 cache sufficiently large such that data written to L2 stays in L2 when it is used.

Chapter 6

Macroblock Scheduling Strategies

Contents

Each scheduling strategy assigns each MB to one of the 4 cores in such a way so that the communication overhead between the 4 cores of the CPU while processing the MBs is minimized. Communication between the cores arises due to the data dependencies between the MBs.
Each MB is also assigned a decoding cycle number.In each decoding cycle at least one and maximum four MBs are processed.Each decoding cycle consists of multiple CPU cycles and the number varies from one decoding cycle to another.

No. of CPU cycles in a decoding cycle = No. of CPU cycles to complete intra core communication
+ No. of CPU cycles to complete processing ,
for each MB processed in that decoding cycle

In the following sections core and decoding cycle assignment of MBs is demonstrated on a 5 by 11 slice containing 55 MBs of the test video.In the examples shown below the dependency graph of luma samples is considered. Another set of core and decoding cycle assignment must

be done taking into account the dependency graph of chroma samples for each slice.

6.1 Greedy Strategy

Firstly decoding cycle number assignment of MBs is done assuming maximum dependency for each MB as shown in Figure 6.1. Each MB can be maximum dependent on 4 other MBs(i.e. the left MB, top right MB, top MB and top left MB of the current MB if present).A MB can be assigned a decoding cycle only after all the 4 MBs on which the current MB is dependent is assigned to some earlier decoding cycles.

Next core assignment of the MBs is done in a greedy manner as shown in Figure 6.2. The MB(1,1) and MB(1,2) is assigned to core 1 without any loss of generality.Thereafter each current MB is assigned to the core which has processed the MB on which the current MB has highest data dependency.If that core is already assigned to another MB in that decoding cycle the next best core is chosen and so on. The MBs to be processed in a particular decoding cycle is assigned a priority based on the amount of data dependency on the 4 neighbouring MBs. The MB with highest dependency on its neighbours has the highest priority. The MBs in a decoding cycle are assigned cores in descending order of priority.

	1	2	3	4	5	6	7	8	9	10	11
1	1	2	3	4	5	6	7	8	10	12	14
2	3	4	5	6	7	8	9	11	13	15	16
3	5	6	7	8	9	11	13	15	16	17	18
4	7	8	9	11	13	15	16	17	18	19	20
5	9	11	13	15	16	17	18	19	20	21	22

Figure 6.1: Decoding cycle number assignment to MBs

	1	2	3	4	5	6	7	8	9	10	11
1	1	1	1	2	2	2	2	2	2	2	2
2	2	1	1	1	1	3	3	2	2	2	2
3	3	3	3	1	1	3	3	3	3	3	3
4	4	4	4	4	4	4	4	1	1	3	3
5	2	1	1	1	1	2	2	2	1	3	3

Figure 6.2: Core assignment to MBs

6.2 Improved Greedy Strategy

A MB may not be dependent on all of its 4 neighbouring MBs as described in earlier section.This strategy takes into account the real dependency of the MBs as described in chapter 2.

Firstly decoding cycle number assignment of MBs is done as shown in Figure 6.3. It is done taking into consideration the dependency graph of the MBs. A ready queue of MBs is maintained. At the beginning of each decoding cycle the slice is scanned to find any MB which has not yet been assigned a decoding cycle number and for which all the MBs on which it is dependent is assigned an earlier decoding cycle number. These MBs are added at the end of the ready queue. 4 MBs are removed from front of ready queue and assigned the current decoding cycle number.This process is repeated for the next decoding cycle until the ready queue is empty or all MBs in the slice has been assigned a decoding cycle number.

Next core assignment of the MBs is done in a greedy manner as shown in Figure 6.4 similar to the strategy described in section 6.1.

	1	2	3	4	5	6	7	8	9	10	11
1	1	2	3	4	5	6	7	8	9	10	11
2	2	3	4	5	6	7	8	9	10	11	12
3	3	4	6	7	8	9	10	11	12	13	14
4	4	5	7	8	10	12	13	14	15	16	15
5	5	6	9	11	12	13	14	16	17	17	18

Figure 6.3: Decoding cycle number assignment to MBs

	1	2	3	4	5	6	7	8	9	10	11
1	1	1	2	2	2	2	2	2	2	2	4
2	2	1	1	4	4	1	1	4	4	2	4
3	3	4	1	4	4	1	1	1	1	1	1
4	3	3	3	3	3	3	3	2	1	1	2
5	1	3	3	3	2	2	3	3	1	2	2

Figure 6.4: Core assignment to MBs

6.3 Vertical + Horizontal Partition

Firstly core assignment of MBs is done as shown in Figure 6.5. The slice is vertically partitioned into two equal halves and then each half is further partioned horizontally into two equal halves. The top-left partition is assigned to core 1. The top-right partition is assigned to core 3. The bottom-left partition is assigned to core 2. The bottom-right partition is assigned to core 4.

Next decoding cycle assignment of the MBs is done as shown in Figure 6.6 assuming each MB dependent on 4 neighbouring MBs. A MB can be assigned a decoding cycle only after all the 4 MBs on which the current MB is dependent is assigned to some earlier decoding cycles.

	1	2	3	4	5	6	7	8	9	10	11
1	1	1	1	1	1	3	3	3	3	3	3
2	2	2	2	2	2	4	4	4	4	4	4
3	1	1	1	1	1	3	3	3	3	3	3
4	2	2	2	2	2	4	4	4	4	4	4
5	1	1	1	1	1	3	3	3	3	3	3

Figure 6.5: Core assignment to MBs

	1	2	3	4	5	6	7	8	9	10	11
1	1	2	3	4	5	6	7	8	9	10	11
2	3	4	5	6	7	8	9	10	11	12	13
3	6	7	8	9	10	12	13	14	15	16	17
4	8	9	10	11	12	14	15	16	17	18	19
5	11	12	13	14	15	18	19	20	21	22	23

Figure 6.6: Decoding cycle number assignment to MBs

6.4 Vertical Partition

Firstly core assignment of MBs is done as shown in Figure 6.7. The slice is vertically partitioned into four equal halves. The leftmost partition is assigned to core 1, second leftmost partition is assigned to core 2, third leftmost partition is assigned to core 3 and rightmost partition is assigned to core 4.

Next decoding cycle assignment of the MBs is done as shown in Figure 6.8 assuming each MB dependent on 4 neighbouring MBs. A MB can be assigned a decoding cycle only after all the 4 MBs on which the current MB is dependent is assigned to some earlier decoding cycles.

	1	2	3	4	5	6	7	8	9	10	11
1		1	1	2	2	2	3	3	3	4	4
2	1	1	1	2	2	2	3	3	3	4	4
3	1	1	1	2	2	2	3	3	3	4	4
4	1	1	1	2	2	2	3	3	3	4	4
5	1	1	1	2	2	2	3	3	3	4	4

Figure 6.7: Core assignment to MBs

6.5 Partition Using Metis

METIS [12] is a collection of serial and parallel programs and libraries that can be used to partitioning unstructured graphs, finite element meshes, and hyper graphs, both on serial as well as on parallel computers. It has been developed by George Karypis, Professor at the Department of Computer Science and Engineering at the University of Minnesota in the Twin Cities of Minneapolis and Saint Paul and a member of the Digital Technology Center (DTC) at the University of Minnesota.

	1	2	3	4	5	6	7	8	9	10	11
1	1	2	3	4	5	6	7	8	9	10	11
2	4	5	6	7	8	9	10	11	12	13	14
3	7	8	9	10	11	12	13	14	15	16	17
4	10	11	12	13	14	15	16	17	18	19	20
5	13	14	15	16	17	18	19	20	21	22	23

Figure 6.8: Decoding cycle number assignment to MBs

METIS can be used to equipartition a graph into n parts such that the sum of weight of the intra-partition edges is minimized. In our problem each partition corresponds to a core and the intra-partition edges correspond to the communication overhead between cores. Using METIS we can partition the dependency graph of a slice into 4 parts such that communication overhead between cores in minimized.

Firstly core assignment is done using METIS as shown in figure 6.9. METIS takes as input the dependency graph of the MBs in a slice and assigns a partition number between 1 and 4 to each MB. The MB is assigned to the core corresponding to that partition.

Next decoding cycle assignment of the MBs is done as shown in figure 6.10 assuming each MB dependent on 4 neighbouring MBs. A MB can be assigned a decoding cycle only after all the 4 MBs on which the current MB is dependent is assigned to some earlier decoding cycles.

	1	2	3	4	5	6	7	8	9	10	11
1	2	2	2	2	2	4	4	4	4	4	4
2	2	2	2	1	1	1	3	3	3	4	4
3	2	2	1	1	1	1	3	3	3	4	4
4	2	2	1	1	1	1	3	3	3	4	4
5	2	2	1	1	1	3	3	3	3	4	4

Figure 6.9: Core assignment to MBs

	1	2	3	4	5	6	7	8	9	10	11
1	1	2	3	4	5	6	7	8	9	10	11
2	6	7	8	9	10	11	12	13	14	15	16
3	9	10	12	13	14	15	16	17	18	19	20
4	11	13	16	17	18	19	20	21	22	23	24
5	14	17	20	21	22	23	24	25	26	27	28

Figure 6.10: Decoding cycle number assignment to MBs

6.6 Partition Using Tuned Metis

The problem with METIS is that it does not take into account the parallelisability of the MBs. It can put independent MBs in the same partition.As a result the scope of parallelisability of the MBs is hampered and they have to be executed sequentially in the same core. To overcome

this problem we create a new input dependency graph given to METIS from the old dependency graph in the following way.

The new dependency graph is a weighted graph $G' = (V', E')$ and the old dependency graph is a weighted graph $G = (V, E)$. Let $w'_{i'j'}$ denote the weight associated with edge $(i', j') \in E'$ where $i', j' \in V'$ and w_{ij} denote the weight associated with the edge $(i, j) \in E$ where $i, j \in V$.

Let

$$w' = median\{w_{ij}\} \forall (i,j) \in E, V' = V, E' = E \cup E_1, \text{ where} \tag{6.1}$$

$E_1 = (i, j)$ such that there is a transitive edge between i and j in G, then

$$w'_{ij} = \alpha \times w_{ij} + (1-\alpha) \times w', if(i,j) \in E = (1-\alpha) \times w', if(i,j) \in E_1 where 0 <= \alpha <= 1 \tag{6.2}$$

METIS is used to partition the new dependency graph G' into 4 parts and core assignment and decoding cycle assignment is done as described in section 6.5 and the result is as shown in figures 6.11 and 6.12 respectively.

The value of α is increased in steps of 0.1 and a core assignment and decoding cycle assignment is calculated for each value of α using METIS. The best schedule among them in terms of speed-up is taken.

	1	2	3	4	5	6	7	8	9	10	11
1	4	4	3	3	3	1	1	1	1	2	2
2	4	4	4	3	3	3	1	1	2	2	2
3	4	4	4	3	3	3	1	1	2	2	2
4	4	4	4	3	3	3	1	1	2	2	2
5	4	4	4	3	3	1	1	1	2	2	2

Figure 6.11: Core assignment to MBs

	1	2	3	4	5	6	7	8	9	10	11
1	1	2	3	4	5	6	7	8	9	10	11
2	3	4	5	6	7	8	10	11	12	13	14
3	6	7	8	9	10	11	12	13	15	16	17
4	9	10	11	12	13	14	15	16	18	19	20
5	12	13	14	15	16	17	18	19	21	22	23

Figure 6.12: Decoding cycle number assignment to MBs

Chapter 7

Implementation, Results and Analysis

Contents

7.1 Setup

- The test video used here is dd_p176x144_intlc_Ionly2̇64.There are 25 frames in the video.Each frame is divided into Top field-Bottom field pairs. Each Top field,Bottom field consists of one slice. Each slice has 55 MBs arranged in 5×11 2-dimensional matrix. There are 50 slices in total.

- The decoder produces a XML file dump for the test video.We have created dependency graphs of luma and chroma samples taking data from the XML file.

- All the experiments are done firstly considering the dependency graph of luma samples and secondly considering the dependency graph of chroma samples

- All graphs have been plotted on data from 50 slices.

- All slices in the test video are I-slices.

7.2 Performance Plots

The Improved Greedy strategy maximizes core utilization as shown in Figure 7.1.

The number of idle CPU cycles spent by each core is inversely proportional to utilization. The Figure 7.2 shows that Improved Greedy strategy minimizes number of idle CPU cycles of each core.

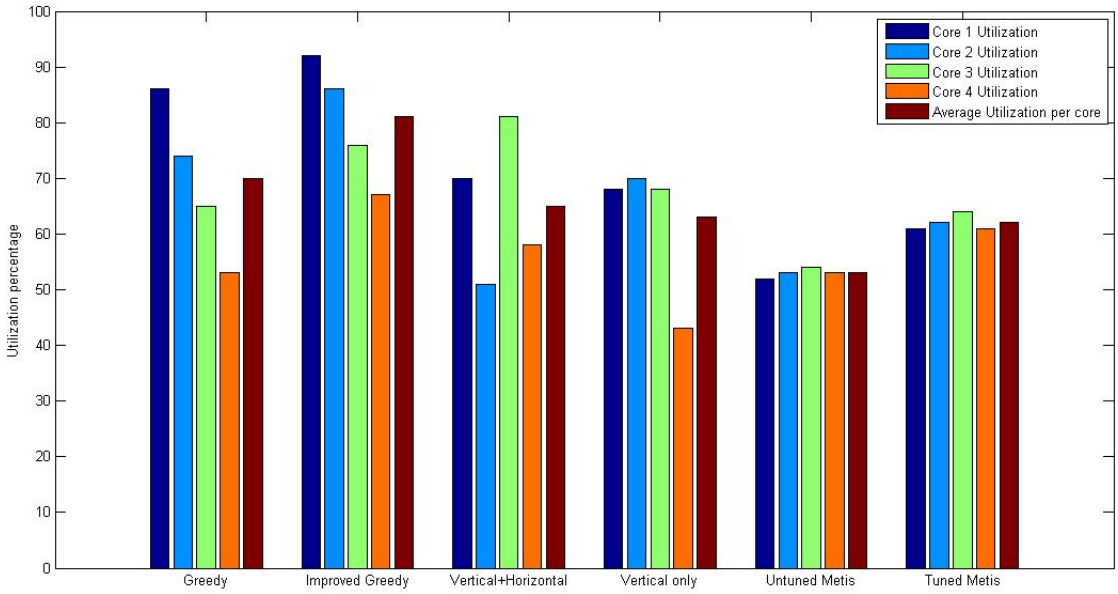
Receiving MB dependent data via L1 cache is cheapest. If a core processes one MB and one of its neighbouring MBs in consecutive decoding cycles then the data required for their dependency is available in the L1 cache of the core. The total latency to read and write data from and to L1 is 4 CPU cycles. The volume of data transferred through L1 caches by each strategy is shown in Figure 7.3. For luma samples the Vertical Only , Untuned and Tuned METIS perform equally well. For chroma samples the Vertical Only strategy performs best.

Receiving MB dependent data via L2 cache is costly. If one MB and one of its neighbouring MBs is processed in consecutive decoding cycles in two cores on the same die then the data required for their dependency is available in the L2 cache of the die. The total latency to read and write data from and to L1 is 30 CPU cycles. The volume of data transferred through L2 caches by each strategy is shown in Figure 7.4. In Vertical+Horizontal strategy the volume is remarkably higher than the other strategy. So this strategy will have huge communication overhead. The Vertical Only , Untuned and Tuned METIS perform equally well as their data volume is lowest.

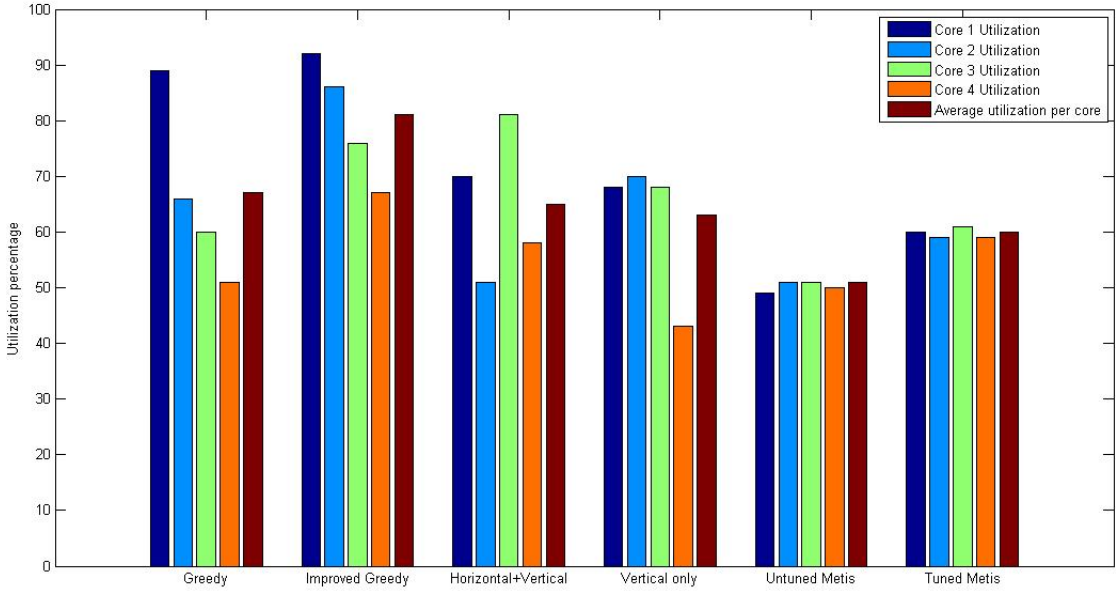
Receiving MB dependent data via L2 cache and Front side bus is costliest. If one MB and one of its neighbouring MBs is processed in consecutive decoding cycles in two cores on different die then the data required for their dependency is transferred via the FSB and L2 cache of the two die. The total latency to read and write data from and to L1 is 33 CPU cycles. The volume of data transferred through L2 caches by each strategy is shown in Figure 7.5. In Greedy and Improved Greedy strategy the volume is remarkably higher than the other strategies. So these strategies will have huge communication overhead. The Vertical Only , Untuned and Tuned METIS perform equally well as their data volume is lowest.

Energy consumption is calculated using the plot shown in Figure 7.6. The average power dissipation is calculated by taking the weighted mean of the number of active cores over all decoding cycles. The energy consumption is calculated by multiplying the average power dissipation with the sum of duration of all decoding cycles.

We can see from Figure 7.7 that Vertical Only consumes minimum energy for luma and Tuned

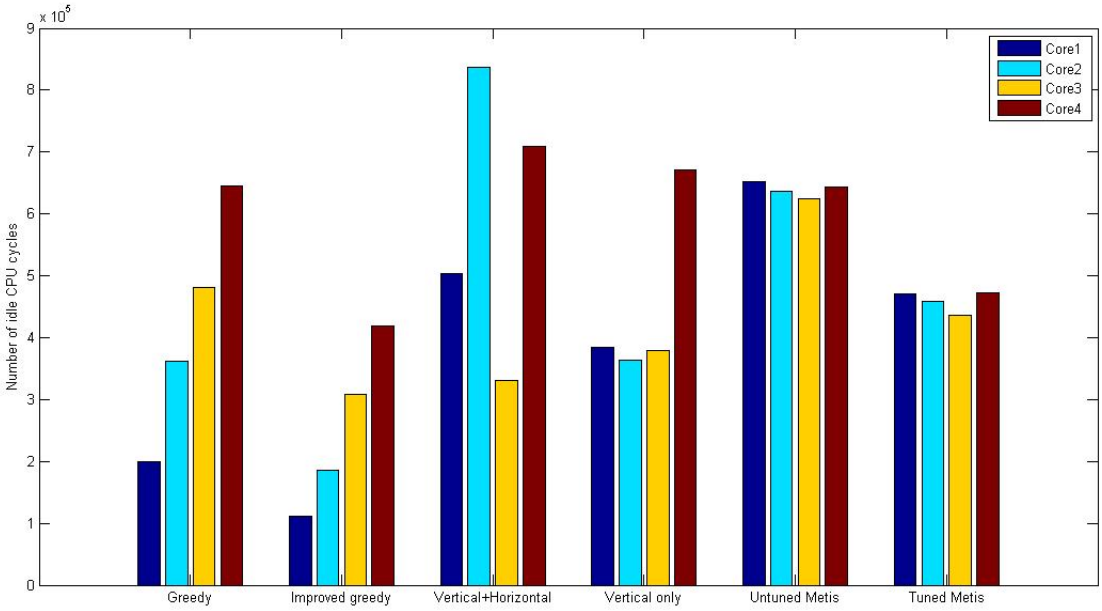


(a) Using luma dependency graph

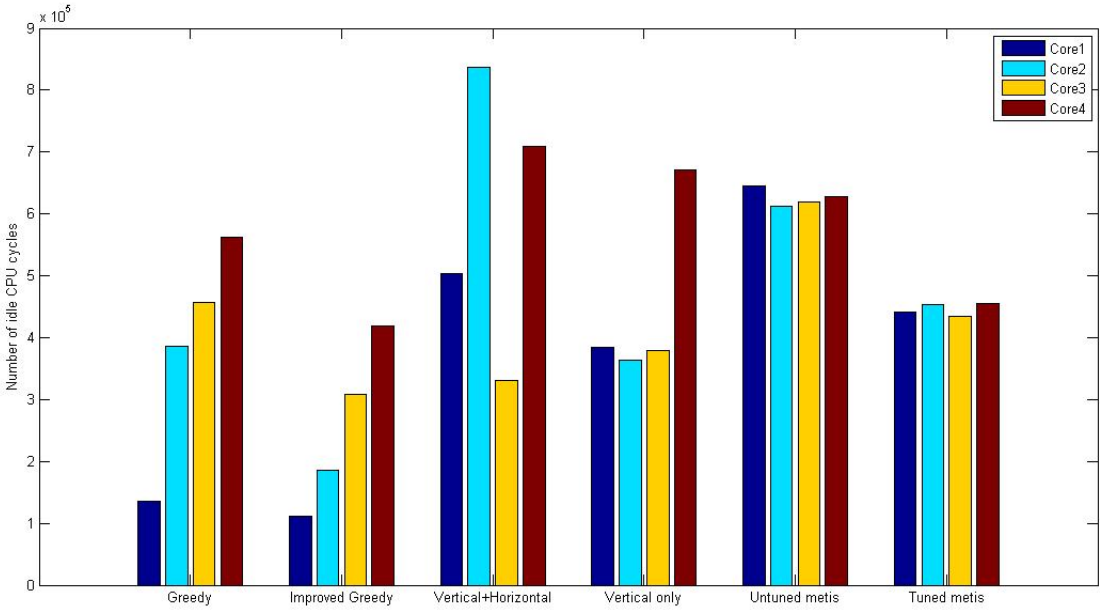


(b) Using chroma dependency graph

Figure 7.1: Core Utilization achieved by each strategy

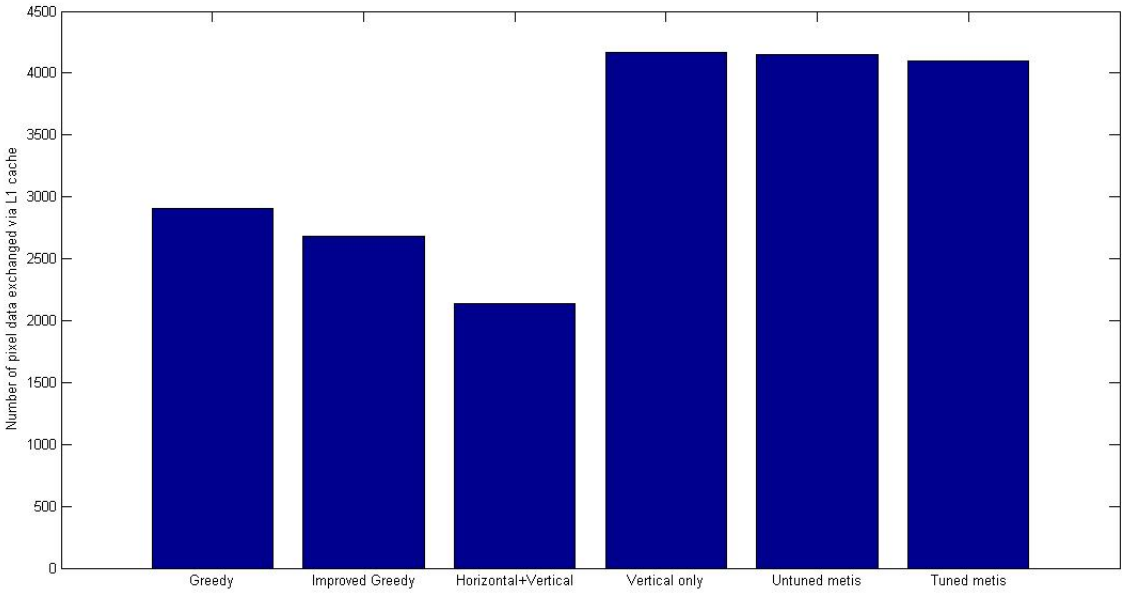


(a) Using luma dependency graph

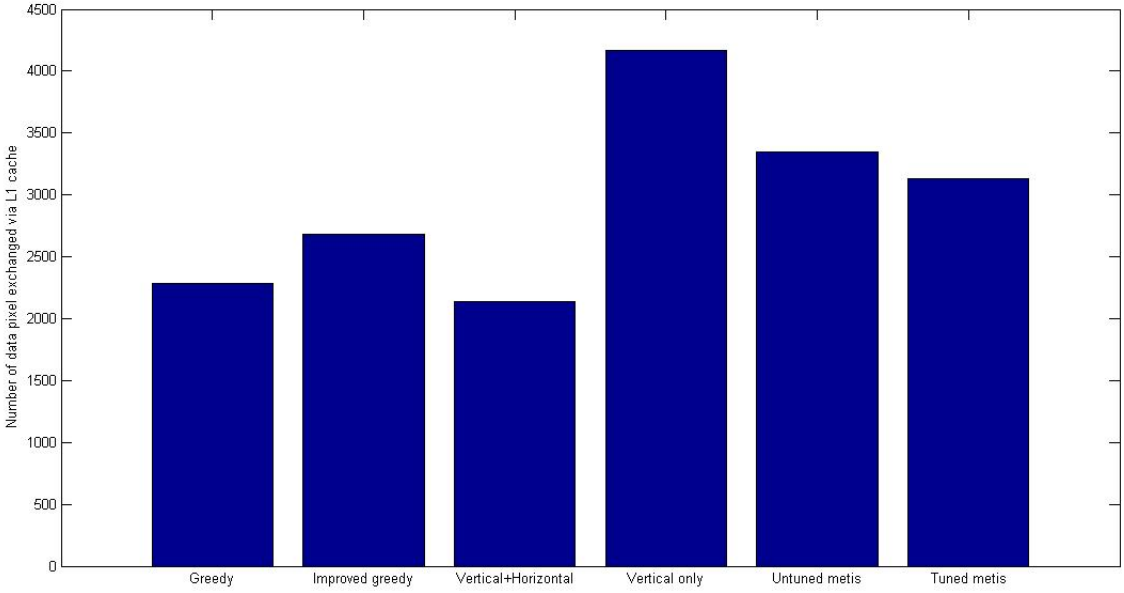


(b) Using chroma dependency graph

Figure 7.2: Idle cycle spent by cores for each strategy

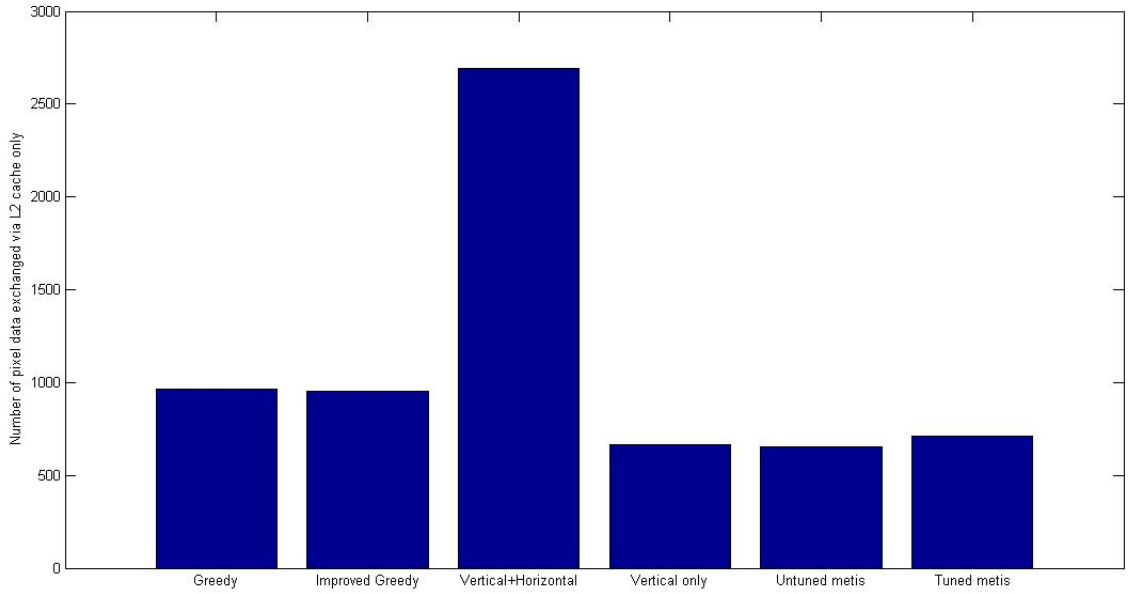


(a) Using luma dependency graph

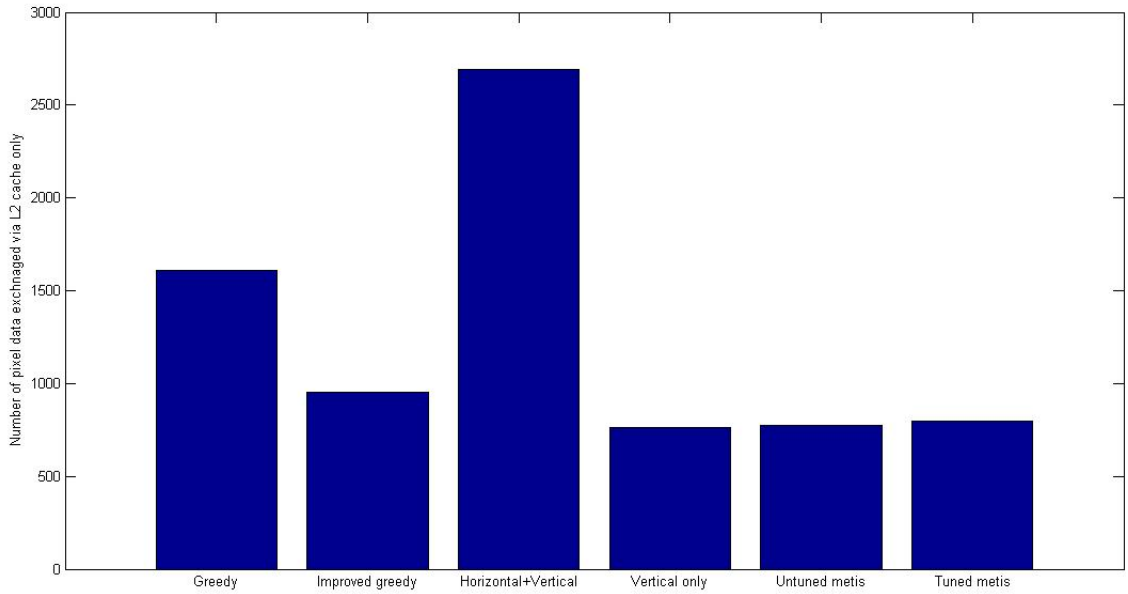


(b) Using chroma dependency graph

Figure 7.3: Pixel data exchange between cores via L1 cache for each strategy

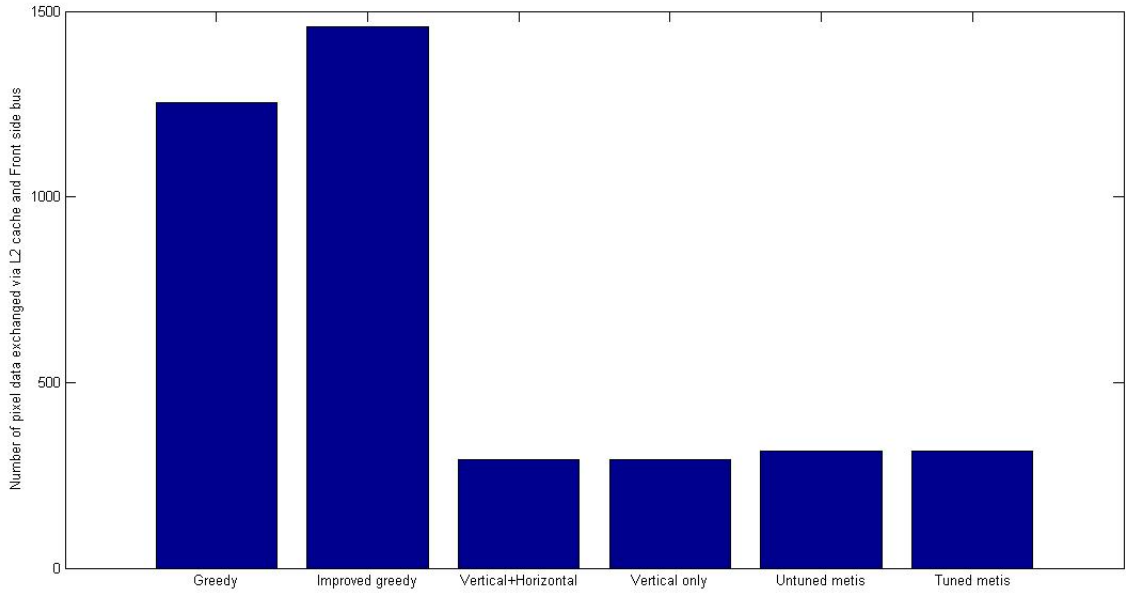


(a) Using luma dependency graph

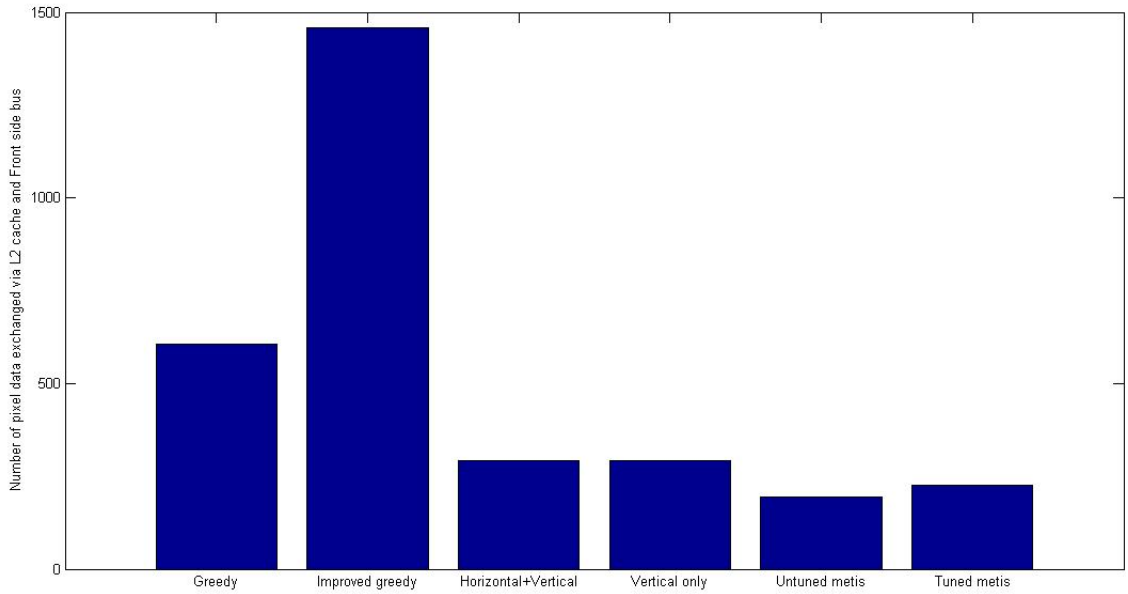


(b) Using chroma dependency graph

Figure 7.4: Pixel data exchange between cores via L2 cache only for each strategy



(a) Using luma dependency graph



(b) Using chroma dependency graph

Figure 7.5: Pixel data exchange between cores via L2 cache and front side bus for each strategy

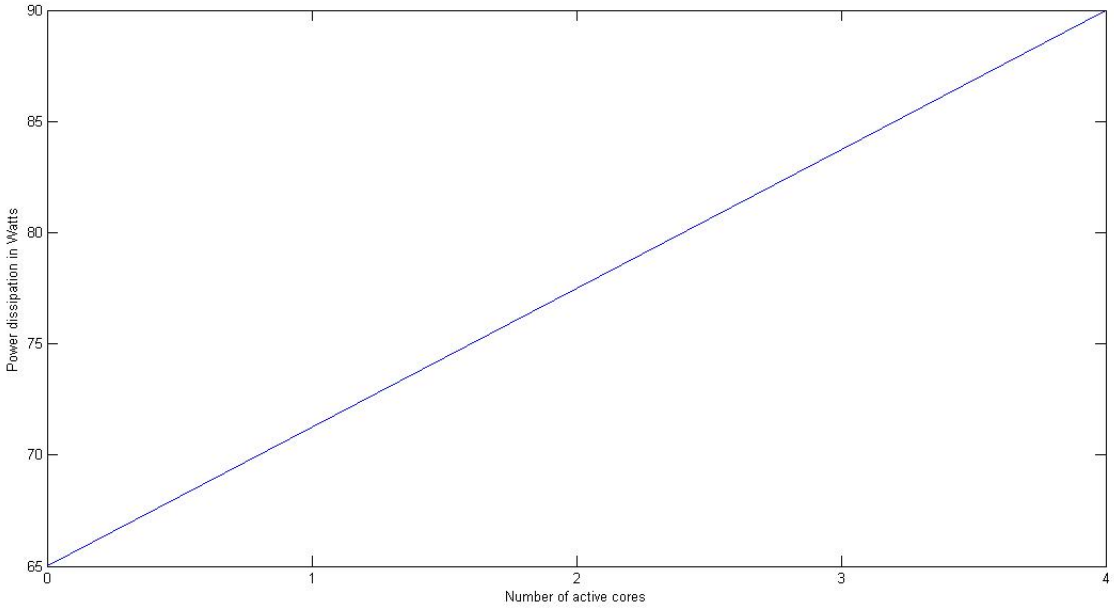


Figure 7.6: Number of active cores vs.Power dissipation plot

METIS for chroma samples respectively.

The Figure 7.8 shows the average duration of decoding cycles of each strategy. The duration of decoding cycle depends on the processing time and communication overhead of all MBs processed in that decoding cycle. Since the processing time of all MBs is more or less same the duration is a function of communication overhead.

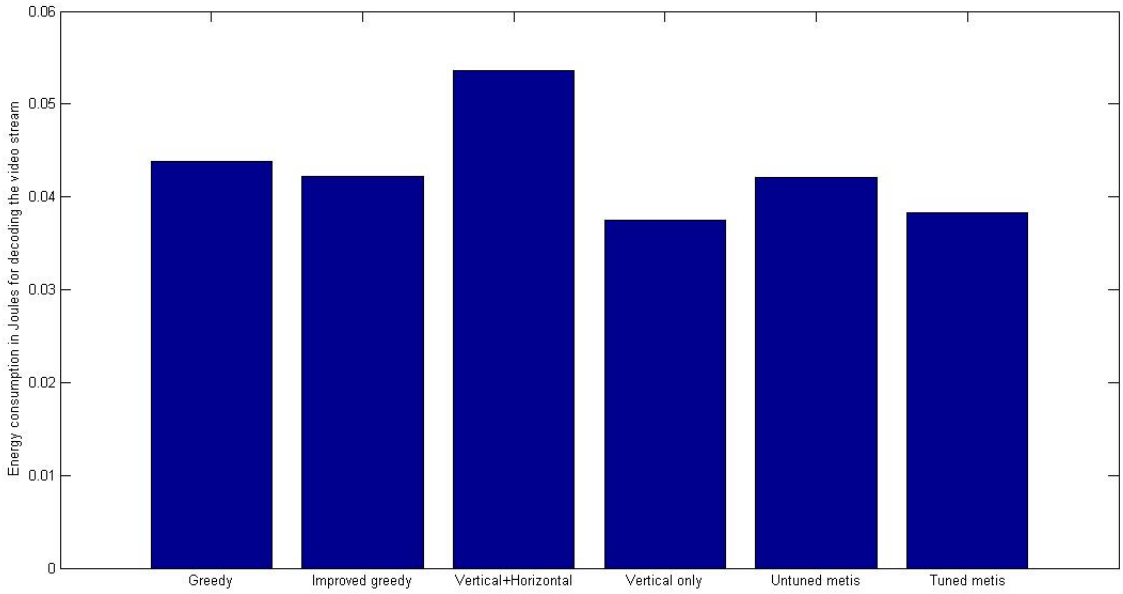
We can see from Figure 7.8 that Tuned and Untuned METIS have minimum average duration of decoding cycles since their communication overhead is minimum as described in Figures 7.4 and 7.5.

The Figure 7.9 shows the number of decoding cycles needed to complete processing of all MBs in the slice by different strategies. We can see Improved Greedy strategy minimizes the number of decoding cycle.

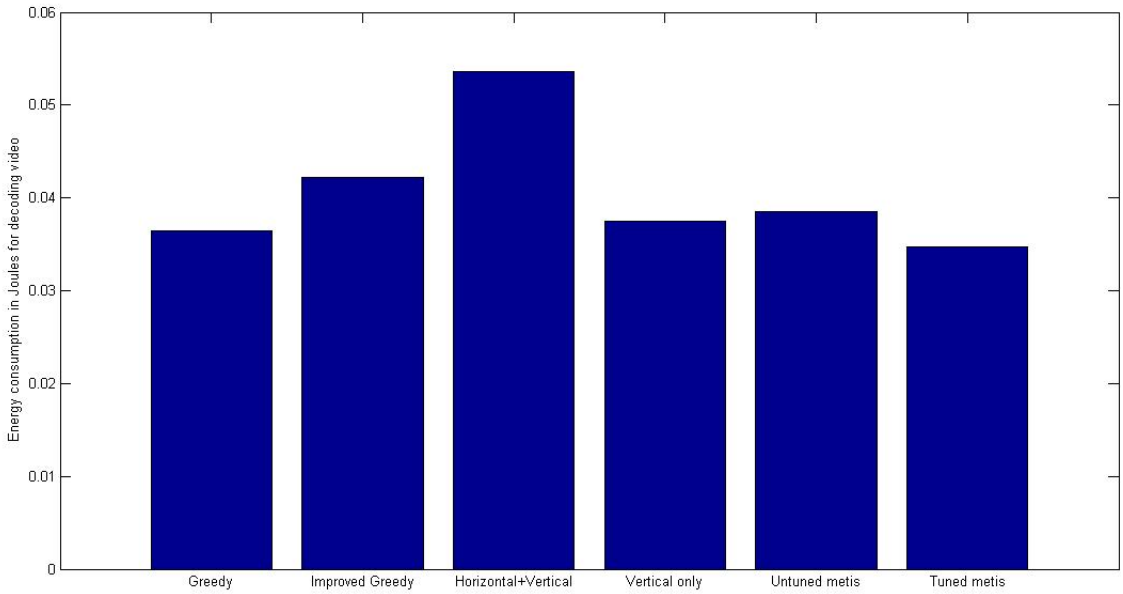
Total processing time of the slice using multi-core = Avg. duration of decoding cycle×No. of decoding cycles (7.1)

$$\text{Speed-up achieved} = \frac{\text{Total processing time of the slice in sequential}}{\text{Total processing time of the slice using multi-core}} \tag{7.2}$$

In some strategies like Improved Greedy the number of decoding cycles is minimum but average duration of decoding cycles is maximum. So their product is not minimum and this strategy



(a) Using luma dependency graph

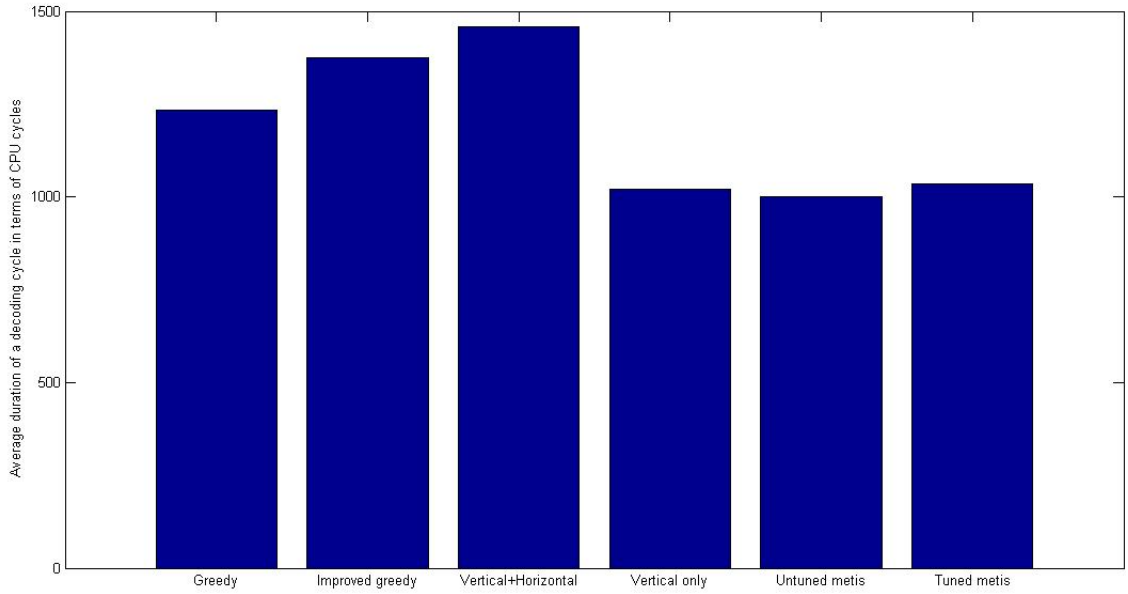


(b) Using chroma dependency graph

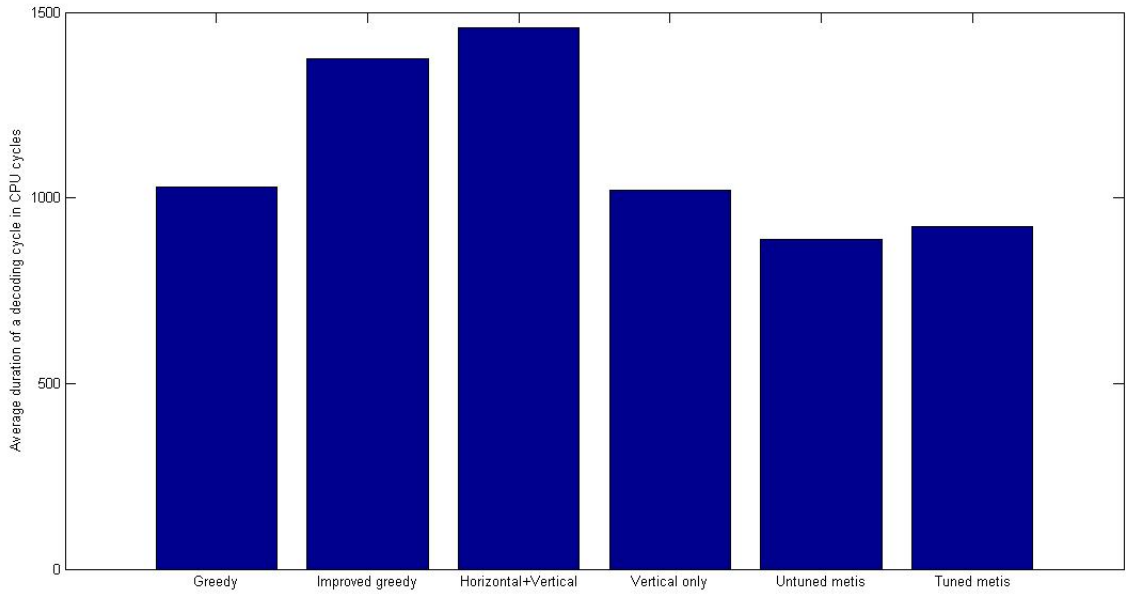
Figure 7.7: Energy consumption to decode video stream for each strategy

does not achieve best speed-up.On the other hand Vertical Partitions Only and Tuned METIS have moderate number of decoding cycles and average duration of decoding cycles. So the total processing time of the slice is minimized by these strategies and they achieve best speed-up as shown in Figure 7.10.

The Tuned METIS strategy Chapter 6.6 is executed for 10 different values of α from 0.1 to 1 in steps of 0.1. The value of α that gives best speed-up varies from slice to slice. The Figure 7.13 shows the values of alpha and the percentage of cases when α gives best speed-up.

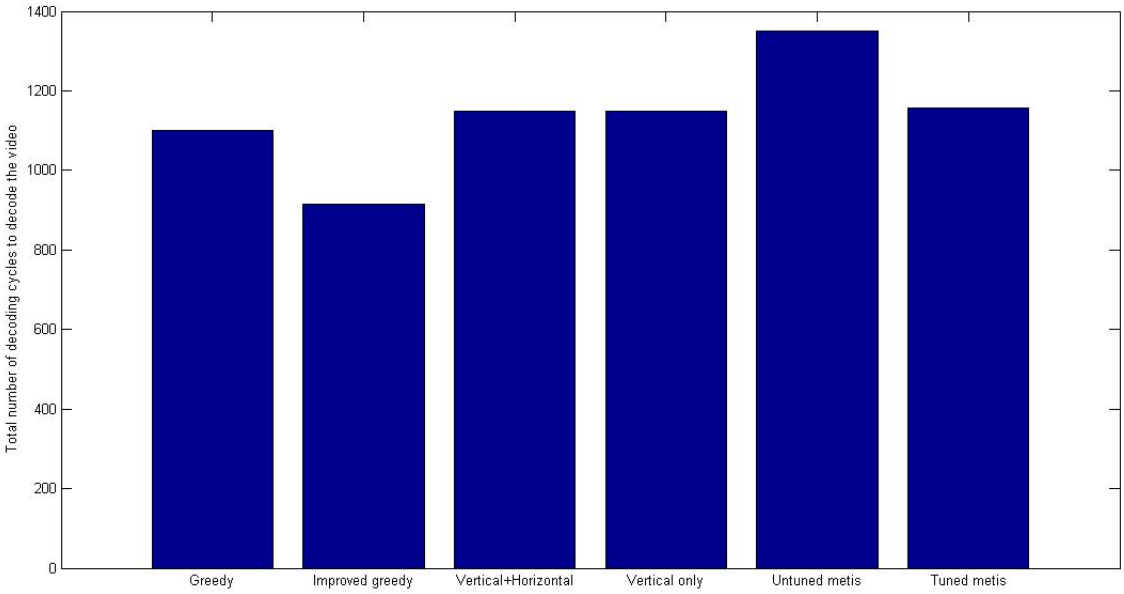


(a) Using luma dependency graph

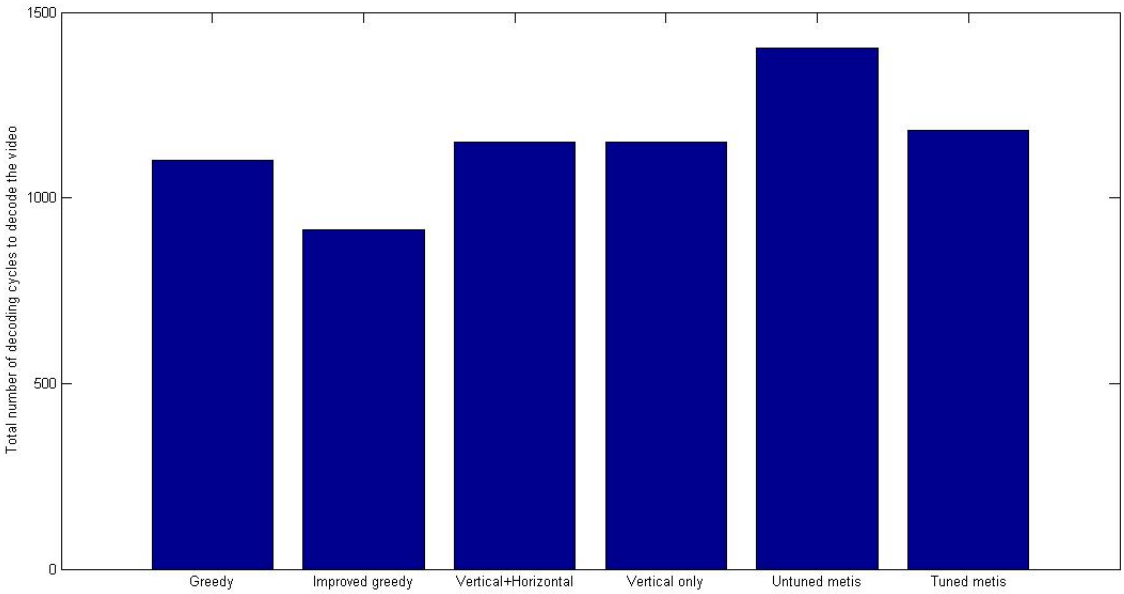


(b) Using chroma dependency graph

Figure 7.8: Average duration of a decoding cycle in terms of CPU cycles for each strategy

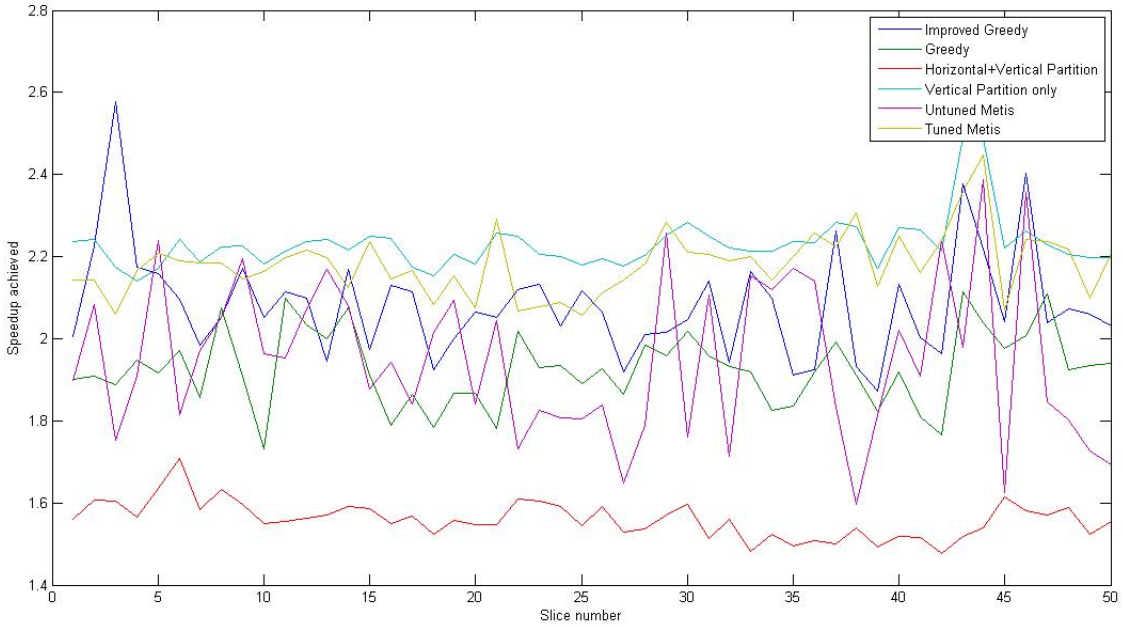


(a) Using luma dependency graph

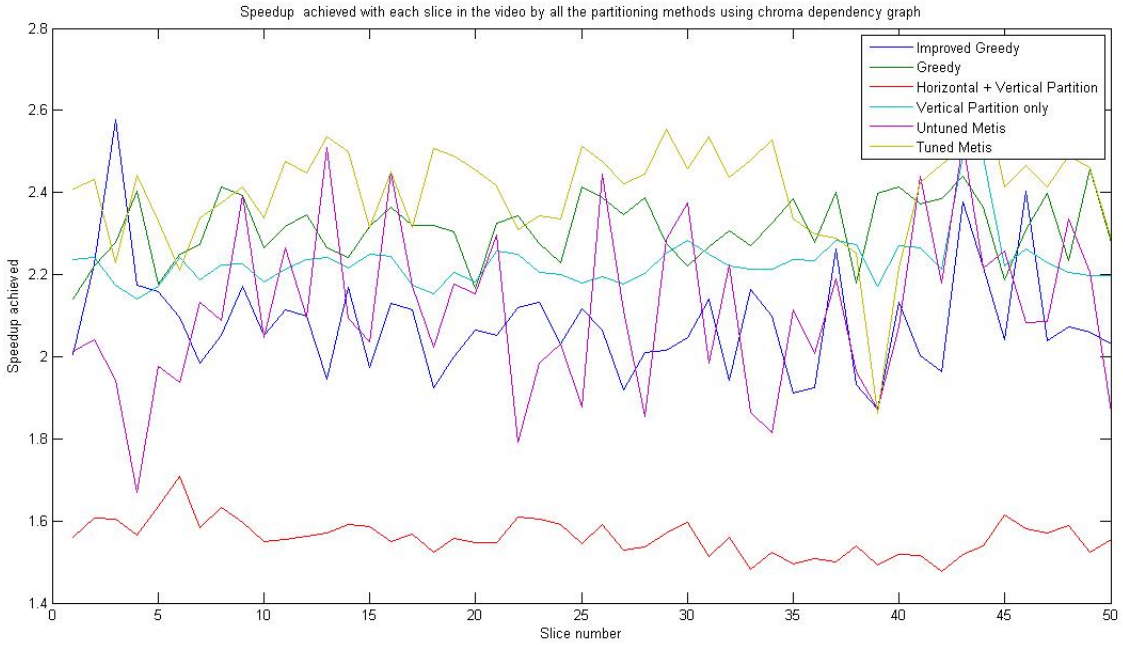


(b) Using chroma dependency graph

Figure 7.9: Total number of decoding cycles to decode the test video by each strategy

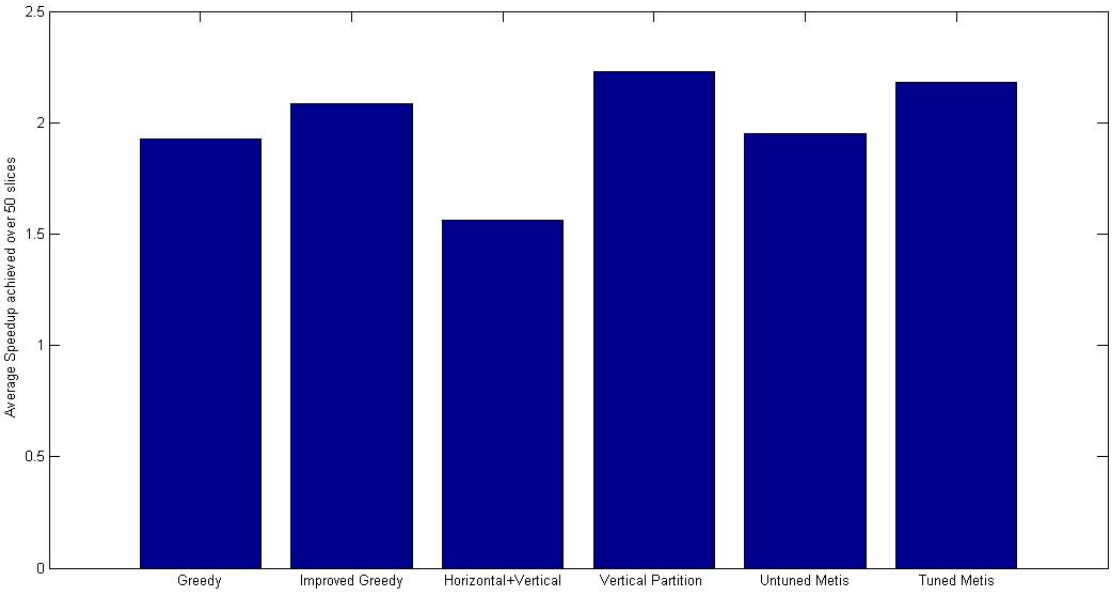


(a) Using luma dependency graph

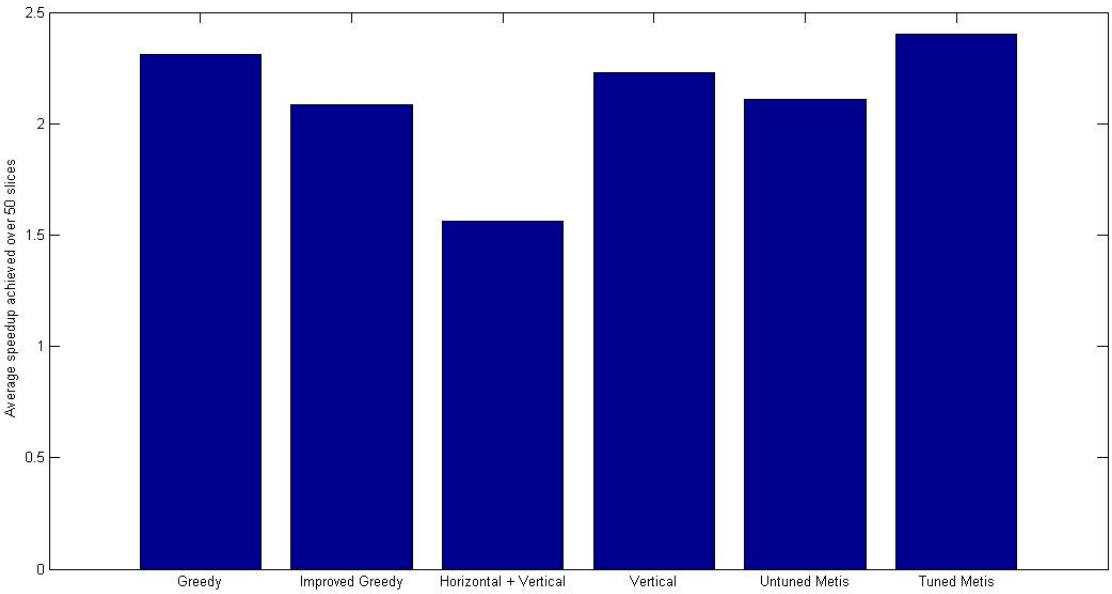


(b) Using chroma dependency graph

Figure 7.10: Speedup vs slice number plot



(a) Using luma dependency graph



(b) Using chroma dependency graph

Figure 7.11: Average speedup achieved by each strategy

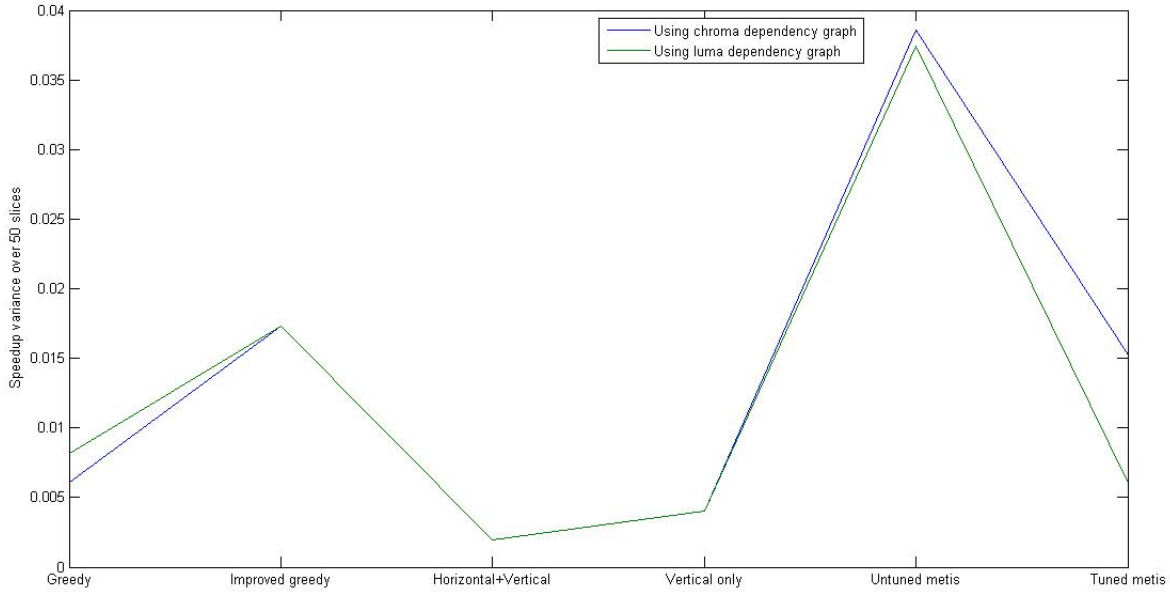
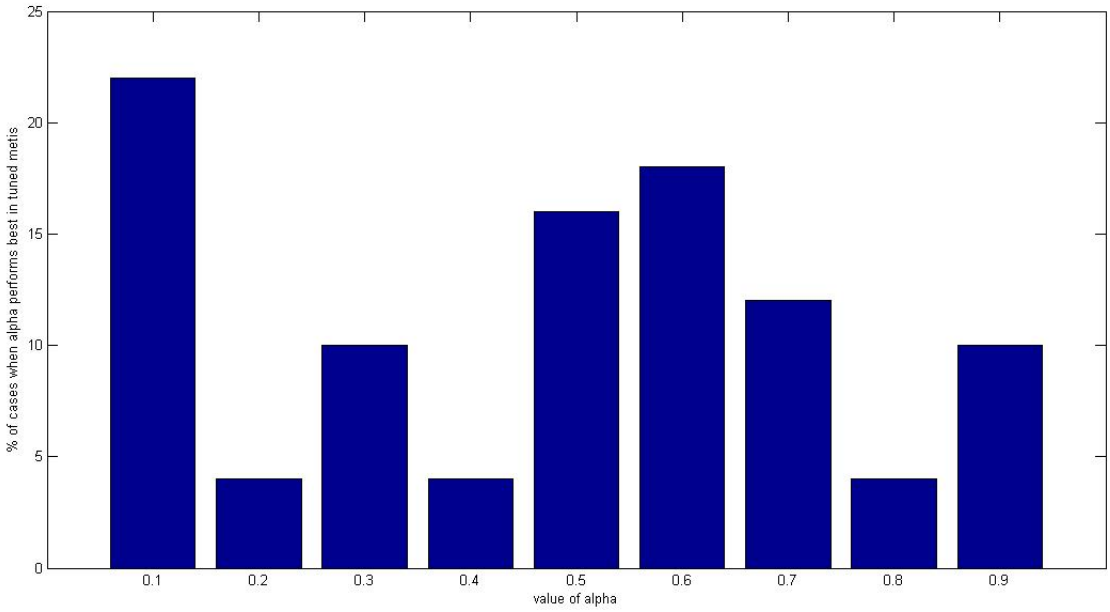
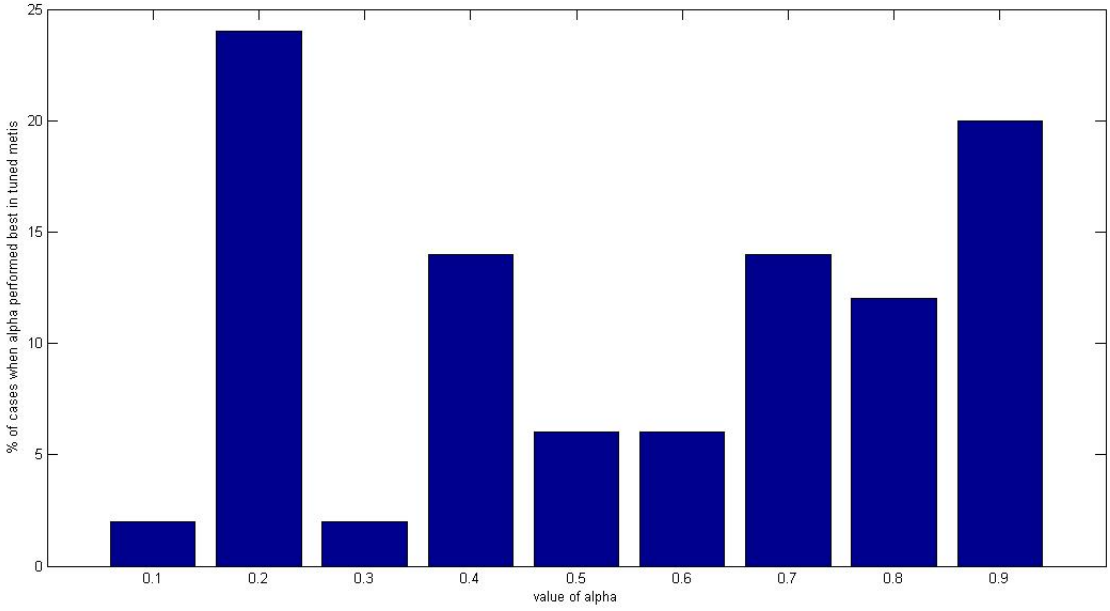


Figure 7.12: Speedup variance over 50 slices of the video for chroma and luma dependency graphs by each strategy



(a) Using luma dependency graph



(b) Using chroma dependency graph

Figure 7.13: Value of α for which Tuned METIS gives best speed-up

Chapter 8

Conclusion and Scope of Future Work

Contents

In this thesis we propose some efficient partitioning strategies of a $H.264$ video slice such that we can assign one partition to one core in a quad-core processor architecture. The overall objective is to minimize communication overhead between cores that arise due to data dependence among the partitions of the slice.

After studying the different performance plots shown in the previous section we can conclude that none of the strategies perform best under all the performance evaluation metrics. One strategy might perform best under one criterion but might perform below expectation under another criterion. So we have to choose a partitioning strategy based on our primary requirement.

If we want to maximize speed-up or minimize energy consumption the Vertical Partition only and Tuned METIS perform best for luma and chroma respectively. To maximize core utilization and minimize idle CPU cycles of the cores Improved greedy strategy performs best. To maximize load balance among the cores Untuned and Tuned METIS perform best. For maximizing consistency in speed-up across all slices in video, i.e to minimize variance in speed-up Vertical

Partition strategy is the best.

The following are some areas in which the work of this thesis can extended to obtain better performance

8.1 Finding Value of α for which Tuned METIS Performance Converges

We have performed experiments to analyse the performance of Tuned METIS by varying the value of α as described in section 6.6 from 0 to 1 in steps of 0.1. Tuned METIS performs best for different values of α for different slice as shown in the plot 7.13. For every slice we have to evaluate performance of tuned METIS for 10 α values. This needs large computation, especially for large slices.

In future experiments could be conducted to find the optimum value of α for which Tuned METIS always performs best and save a lot of computation time.

8.2 Implementing the Strategies for Larger Videos

8.2.1 Introducing L1/L2 Cache Replacement Policies

The slice partitioning strategies discussed in the previous sections performs good for small videos that have slice size(for e.g. 6 by 12). In our thesis we have not considered the L1/L2 cache replacement policies. We have assumed that L1 cache is large enough to hold MBs that have been processed in a core in the last 4 decoding cycles.

For large videos we have to take into consideration the cache replacement policy. We can partition large slices in large videos into multiple smaller tiles of the size say 6×12. Each small tile can be then processed using the strategies discussed in the previous sections. We have to also take into account communication overhead due to data flow across tile boundaries.

8.2.2 Using Hamming Distance for Dependency Graph Generation in Tuned METIS

For tuned METIS strategy in large videos the modified dependency graph containing transitive dependence edges become huge. A lot of work can be saved if transitive edges are introduced between the current MB and any MB within a Hamming distance k from the current MB such

that performance is not degraded. This can greatly reduce computation time. Finding an optimum value of k remains yet another research area.

Bibliography

[1] Ian E. G. Richardson,*H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia*, WILEY.

[2] Jae-Beom Lee, Hari Kalva,*The VC-1 and H.264 Video Compression Standards for Broadband Video Services*, Springer.

[3] ITU-T Rec. H.264, Version Mar. 2010, *Advanced Video Coding for Generic Audiovisual Services.*

[4] Meenderinck, C., Azevedo, A., Alvarez, M., Juurlink, B., Ramirez, A.:" Parallel Scalability of H.264." In: Proc. First Workshop on Programmability Issues for Multi-Core Computers. (January 2008).

[5] A. Azevedo, B.H.H. Juurlink, C.H. Meenderinck, A. Terechko, J. Hoogerbrugge, M. Alvarez, A. Ramirez, M. Valero, "A Highly Scalable Parallel Implementation of H.264," Transactions on HighPerformance Embedded Architectures and Compilers (HiPEAC), September 2009.

[6] A. Azevedo, C. Meenderinck, B. Juurlink, A. Terechko, J.Hoogerbrugge, M. Alvarez, A. Ramirez, and M. Valero, "Parallel H. 264 Decoding on an Embedded Multicore Processor," *Proceedings of the 4th International Conference on High Performance Embedded Architectures and Compilers*, Springer, 2008, pp. 404-418.

[7] Jike Chong, N. Satish, B. Catanzaro, K. Ravindran, and K. Keutzer,"Efficient Parallelization of H.264 Decoding with Macro Block Level Scheduling," *Multimedia and Expo, 2007 IEEE International Conference on,* 2007, pp. 1874-1877.

[8] Shuwei Sun, Dong Wang, and Shuming Chen, "A highly efficient parallel algorithm for H.264 encoder based on macro-block region partition," *Lecture Notes In Computer Science* 2007, pp. 577-585.

[9] Sihn, K., Baik, H., Kim, J., Bae, S., Song, H., "Novel approaches to parallel H. 264 decoder on symmetric multicore systems".*Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE Computer Society* 2009, pp. 2017-2020.

[10] Ahmet Grhanl, Charlie Chung-Ping Chen, Shih-Hao Hung, "GOP-Level Parallelization of the H.264 Decoder without a Start-Code Scanner," *2nd International Conference on Signal Processing Systems (ICSPS),* 2010, pp. V3-627-V3-630.

[11] Mauricio Alvarez Mesa, Alex Ramirez, Arnaldo Azevedo, Cor Meenderinck, Ben Juurlink, Mateo Valero, "Scalability of Macroblock-level Parallelism for H.264 Decoding," icpads, pp. 236-243, 2009 15th International Conference on Parallel and Distributed Systems, 2009.

[12] http://glaros.dtc.umn.edu/gkhome/views/metis