Scalable process network based application modeling for multiprocessors

Abstract

Kahn process network (KPN) is a model of computation used in modeling signal processing and media applications. The application is split into concurrently executing tasks and the communication is made explicit in a KPN model and hence it is suitable for implementation using multiprocessor. This paper describes the techniques for modeling applications using KPN for efficient execution on multiprocessor platform. We propose the scheme of scalable KPN models using the concept of data dependency graph which enables easy exploration of data parallelism in any application modeled using KPN. An MPEG-2 Encoder was taken as case study and modeled as a Kahn process network and various optimization techniques suggested in this paper were applied on that model. The model was simulated on a multiprocessor platform and shows good scalability and performance compared to the base model.

1 Introduction

Rapid advancement in VLSI technology has made the concept of high performance multiprocessor on a single chip a feasible and attractive solution for many applications. Other than scientific computing common applications are large database servers, graphics rendering processors, video servers, set-top boxes, video games and multimedia based content delivery systems.

One main bottleneck in widespread adaptation of multiprocessor is the difficulty in programming it. In most of the approaches [1, 2, 3] the application modeling is tightly coupled with the final target implementation. Such low-level abstract programming results in limited portability of the model across different architectures. Also, current SoC multiprocessors[4] are heterogeneous in nature with both hardware and software component. Thus parallel programming model should be implementable in both hardware & software system. Hence it is absolutely essential to hide all the architecture details during the modeling stage for implementation on SoC multiprocessors. KPN model is an abstract model independent of the underlying architecture which could be either implemented as complete hardware or as tasks in multiprocessor. Thus application can be modeled and evaluated separately and the mapping on the target architecture (namely multiprocessor) could be done at later stages[5].

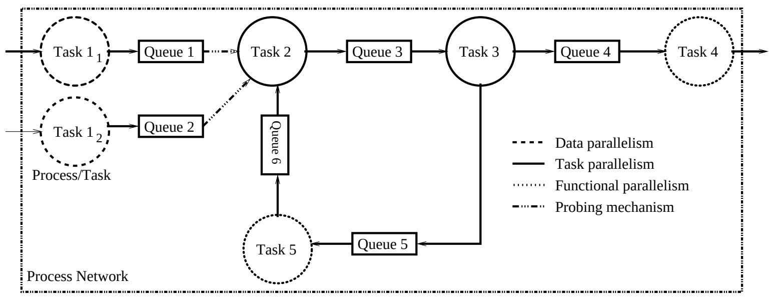

Figure 1: A sample KPN model

Kahn Process Network (KPN)[6, 7] offers a convenient method for modeling of stream based applications. KPN model partitions an application into concurrently executing tasks and hence implementing KPN on multiprocessor is an attractive solution. Even though KPN captures the task level parallelism available in any application, data parallelism has to be explored manually and intuitively. This is a serious limitation of the KPN model and has been addressed in this paper and scalable KPN model for 3D Recursive Search(3DRS) block matching algorithm (BMA) for motion estimation was derived. Earlier parallelization techniques were reported[1, 8, 9] only for simple motion estimation algorithm and not for 3DRS-BMA. In this paper we also discuss modeling optimization which has to be carried out in the KPN model for efficient execution on multiprocessors. An MPEG-2 decoder was implemented using KPN model as the framework in [10] but data parallelism was applied on a limited scale.

2 Background

Kahn process network (KPN) model consists of number of concurrently executing processes[1] communicating via point-to-point unbounded FIFO channels (Fig.1). KPN explicitly models task level parallelism (or pipelining) and inter-task communication in the application. Thus, it is easy to map KPN model onto any multiprocessor architecture. Kahn[6] has proved that these networks are deterministic i.e they exhibit the same input/output behavior irrespective of their scheduling. An application modeled using KPN can exhibit task, data and functional parallelism. *Functional parallelism* exist in a model, if same input data is processed by two or more different processes. *Data parallelism* exist if different data is processed by different instance of same process at any instant. An example of a KPN model with all types of parallelism is shown in Fig.1. Any KPN model exhibits task level parallelism but exploring the application for data and functional parallelism is non-trivial. This paper addresses the issue of identifying the data parallelism in any application(Sec.5) modeled using KPN.

In order to simulate the behavior of KPN, the C++ class library called YAPI[5, 7] was used. Throughout the paper we denote the tool as YAPI (Y-chart Application Programmers Interface) and the model as KPN. A KPN process blocks if no data is available for reading from its input FIFO (read blocking). If all the processes in a KPN

[1]the terms process, task and thread are similar in KPN

model blocks on read, it is termed real deadlock. KPN model assumes FIFO of infinite size. As it is practically not possible to allocate FIFO of unbounded size, YAPI restricts the FIFO size. If a process attempts to write into a full FIFO it gets write blocked. This inherently results in an artificial deadlock where atleast one process is write blocked[7]. YAPI provides a mechanism to detect these artificial deadlocks and adjust the FIFO sizes accordingly.

Reactivity is an important concept for application modeling but KPN does not support this. Reactivity is present, when we might not be able to predict the sequence of operation which will be performed on user input data. Also, during data parallelism we cannot predict the execution time for each processes (which is data dependent). In these cases, YAPI provides mechanism to check FIFOs for availability of data (called *probing*) and then select a particular FIFO before a read or write operation. This mechanism helps in modeling reactivity and results in a non-deterministic behavior dependening upon the execution sequence of KPN processes. Hence YAPI simulates a bounded FIFO, deterministic and non-deterministic KPN model.

3 Modeling Techniques

Once an application is broken into concurrent processes, several techniques need to be applied for efficient execution of a KPN model on multiprocessor platform:
1. Processes granularity:
Execution time of a model on a multiprocessor depends on the total communication and also on the number of parallel process. A model with fine grain processes shows improved parallelism but at the cost of increased context switching, communication and congestion. On the other hand a model with few course grain process shows less communication, congestion and context switching also has less parallelism. The granularity of processes thus effects the performance of model on multiprocessors. Attempts should be made to break the KPN model for more parallelism but without changing the total tokens transferred for improving the performance on multiprocessor.
2. Scalability:
Every process developed in the base model has to be explored for improved parallelism and the whole model has to be refined to scale easily. A scalable model ensures that given architecture resources are effectively utilized. The scalability directly effects the amount of parallelism, congestion in communication and context switching. The need for scalable application model for multiprocessor is because of the following:

a) KPN modelling is independent of final architecture, hence fine tuning during architecture mapping to efficiently utilize the resources (processors) would be possible only if the model is scalable.

b) When using the model to build a bigger application it would be necessary to partition the resources and hence the model need to be scaled down to prevent resource conflict. In this case a static model would not allow this flexibility and hence the final implementation would not exhibit good performance, which would require redesigning and/or modification of the entire model.

c) Scalable models allow fine tuning and grouping of process to trade-off between communication and computation costs. When these models are needed for application where QoS is of main consideration, scalability would offer sufficient leverage to fine tune the model for specified QoS constraints.

3. Uniform Load Characteristics:

If one or two processes have very high computational time compared to other processes in the model then they dominate the total execution time. These processes can be called as *bottleneck* processes. Such models are clearly inefficient because beyond two processors, the model will not show any speed-up for more processors. If the whole model has more or less uniform execution time for each process, it would be efficient for multiprocessor.

4. Minimum handshake between Processes:

In handshake based implementation effectively only few processes will be performing the job while others wait for response. Since after each computation the sender process puts a request and waits for response from the other process resulting in a context switch for each computation. As demonstrated in [10], handshake based KPN implementation would reduce the effective parallelism present in the model and in turn would increase the number of context switches reducing the streaming behavior of the application. Also, number of tokens transferred between processes for handshaking increases the communication load. Thus KPN model should be more uni-directional with less handshake.

5. Shared memory communication:

One main issue at the modeling stage is, how to utilize shared memory available in multiprocessor architectures. Current YAPI does not have explicit constructs to show that two or more processes are communicating by a shared memory. One possible solutions is to use pointer communication. But this makes the entire model non-portable to other architectures. Another possible solution is introduce a new shared memory channel between the processes, along with the FIFO channels.

6. Optimum buffer size:

In a multiprocessor architecture every data written into a FIFO is converted into a semaphore or snoop operation . We can also group a number of write and perform a single vector write operation which has only one semaphore or one snoop operation. If larger vector write is performed to reduce semaphore operations, it results in increased FIFO memory usage, deadlocks and also increased cache miss. So selecting an optimum size of data transfer between process is essential to trade-off communication load and cache misses / memory size.

4 A Base MPEG-2 Encoder model

MPEG-2 encoder was taken as a case to study the behavior and performance of KPN models on multiprocessor platform. Direct mapping of important blocks suggested in [11],[12],[13] as KPN tasks would give a MPEG-2 KPN encoder model shown in Fig.2. We call this model as the base model[14]. The bubbles indicate processes or tasks and the arcs indicate the FIFO. The most important part of this model is the centralized anchor frame memory manager. The motion estimator process (**Tme**) finds out the

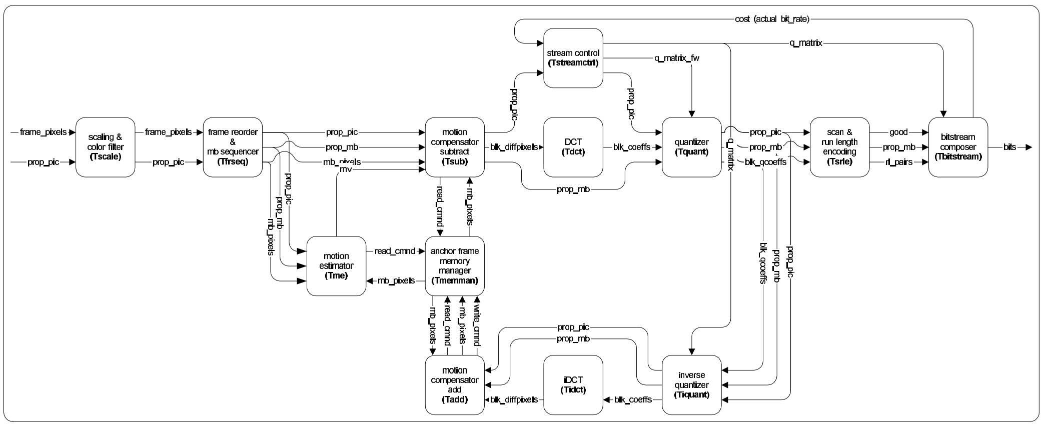

Figure 2: Base Encoder Model

best matching macroblock (MB) in the reference frame, for the input MB. The motion compensated subtract process (**Tsub**) decides the type of encoding for a B/P picture (intra, forward, backward, interpolated). Also **Tsub** process determines the quantisation factor which has to be applied for the MB. The motion compensated add process(**Tadd**) reconstructs the I/P frame, which will be the reference for the next B/P picture. The anchor frame memory manager (**Tmemman**) stores the reference frame in the memory and returns the suitable macroblock requested by the **Tsub**,**Tadd** and **Tme** process. The remaining processes are self explanatory. Among all the tasks, the DCT/IDCT was executing for more than 60% of the time. Because of the execution complexity, regular structure and efficient algorithms, DCT/IDCT task were implemented as hardware tasks. The motion estimation(**Tme**) is the next dominant task interms on execution time. The remaining part of the paper discusses, scalable implementation of motion estimator and the transformations in the model with properties suggested in Sec.3 for efficient execution on multiprocessors.

4.1 3DRS Motion Estimation

In our MPEG-2 encoder model, motion estimation was implemented by 3DRS algorithm[15]. In MPEG encoding, spatial macro-blocks(MB) are MBs which lie in the current frame (Fig.3) and temporal MBs are MBs which belong to its reference frame. A *candidate MB* is defined as the MB whose already calculated motion vector is used by the current MB for block matching. 3D Recursive search (3DRS) is an progressive approach, which uses information from spatial and temporal neighborhood candidate MBs to calculate the motion vector for the current MB[15]. In the original implementation, [15] each MB had 2 spatial candidates and 2 temporal candidates (called as convergence accelerator) and one zero motion vector. The fifth candidate was added to improve the accuracy of motion vector and was not present in the original 3DRS algorithm.

3DRS algorithm has good performance and less complexity over all the existing motion estimation algorithm[15]. Fig.4 shows the comparison of the encoding time for

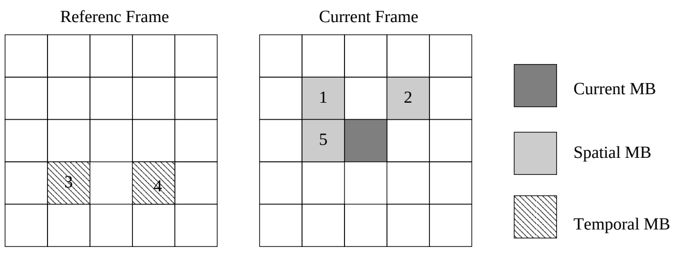

Figure 3: 3DRS motion estimation

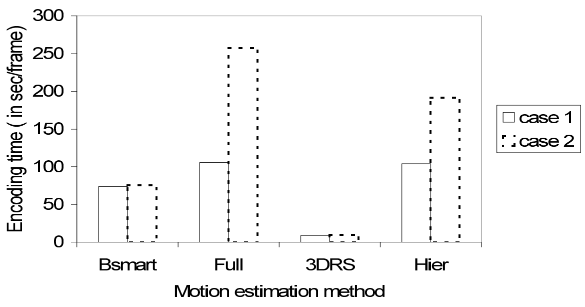

Figure 4: Comparison of Block motion estimation algorithm on the target architecture

different motion estimation algorithms. Also as seen in the Fig.4-case 1, 3DRS is atleast 6-10 faster than other algorithms. The *prediction distance* is defined as the distance (in frames) between the current frame and the reference frame. As the current frame moves away from the reference frame(as prediction distance increases) the search region also increases in other algorithms and so the encoding time increases significantly (Fig.4-case2). In contrast, the 3DRS algorithms encoding time is almost independent of the prediction distance. Full-search algorithms encoding efficiency was on an average only 5% better than 3DRS algorithm. Even though 3DRS motion estimator is a promising candidate for real-time implementation the algorithm is not fully scalable unlike full-search and hierarchical search algorithms[1, 8]. This is because, the motion estimation(ME) for the current macro block is dependent on the ME of other blocks. The main challenge is to model a partially scalable 3DRS motion estimator in the MPEG-2 model which shows better performance on multiprocessors.

5 Scalable KPN model

The Data dependency graph(DDG) is useful to find out the maximum parallelism possible for each process in a KPN model. It shows the data dependency for execution of a process for sequence of inputs applied.

Representation:A DDG for a process consists of edges and nodes. A node represents an instance of process execution and the edges represent the dependency between the execution of that process and execution of another instance of the same process but at a different time (different input). An edge exist in a DDG if some data dependency exists between two different execution of a particular process. A DDG for 3DRS motion estimator is shown in Fig.6.

5.1 Scheduling

The DDG by itself does not explain the details of possible parallelism in the application. The technique of *scheduling* alters the DDG such that all tasks which are independent and which exhibit data parallelism are easily identified and the execution order determined. The method of scheduling a DDG is similar to the scheduling of a data-flow graph (DFG) in high level synthesis of digital circuit. A particular node of a DDG can be scheduled to operate at a slot, if all the inputs for that DDG is available. The KPN FIFO reacts immediately on availability of data in its empty input FIFO, so we need to adopt ASAP (As Soon As Possible) algorithm[16] to find the execution sequence of DDG. The final scheduled graph would indicate the maximum parallelism possible, execution schedule and also minimum time needed for the execution of a particular input with infinite resources. The main advantages of scheduled DDG (SDDG) are:
1. By using SDDG, we can find the scalability of a particular algorithm and trade-off to select an algorithm depending on the available resources.
2. Determine the maximum scalability and speed-up which is possible for an algorithm.
3. Also the amount of resources needed for achieving a particular parallelism can be found out. In fact using the SDDG we can easily extend the KPN model and explore for data parallelism for more complex processes. Constructing such a SDDG helps in building scalable model.
4. Construction of scalable hierarchical KPN model is simple with DDG. If a group of consecutive (adjacent) processes has the same DDG (isomorphic), then they can be grouped into a single process network having the same DDG structure.

5.2 Data dependency graph for ME algorithms

1. DDG of Full search (FS) ME
As it is clear from Fig.5 the DDG does not have any edge between the nodes of ME. This is because the ME for one MB is *independent* of the motion estimation of the other MB. If $M \times N$ MB exists in a frame, it is possible to schedule all the $M \times N$ MB in a single time slot, which means if we have $M \times N$ processors, all can operate concurrently to encode a MB separately to complete the entire motion estimation in a single time slot. Even though, ME using FS can achieve full scalability, the complexity of FS is very high and it requires large number of candidates for motion estimation hence not feasible for real-time implementation.

The concurrency is determined by *Parallelism factor*(P), It indicates the maximum number of node\process which can exist\execute in a single time slot.

$$P(Parallelism factor) = M \times N \tag{1}$$

2. DDG of ME using 3DRS search
From the description of the 3DRS algorithm it looks like parallelisation is not possible

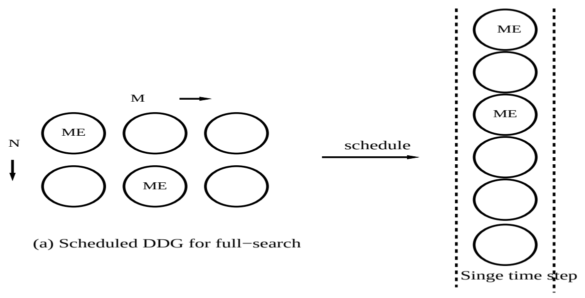

Figure 5: DDG and SDDG of Full-search ME

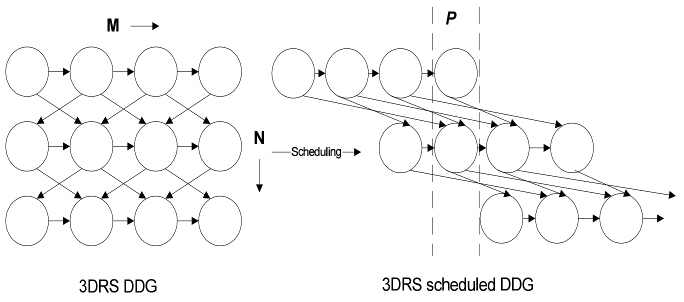

Figure 6: DDG and SDDG of 3DRS ME

using the data parallel approach because of the recursive dependency for ME of a particular MB. But from Fig.6 we can see that it is possible to concurrently execute some ME of different MB at a single time-slot after scheduling. The P factor for 3DRS with candidates shown in Fig.3 for a picture consisting of M × N macro-block is

$$P(Parallelismfactor) = min\left(N, \left\lceil\frac{M}{2}\right\rceil\right) \tag{2}$$

The maximum parallelisation(P) possible is 23 for 720×588 frame(16×16 macro-block size). If we apply the original 3DRS candidates list[15], then the P factor was found out to be 45. The reduced P factor in new model is because of the candidate 5 in Fig.3 which reduces the effective parallelisation within a slice. The KPN model of the scalable 3DRS motion estimator is shown in Fig.7. This figure also indicates the Pel partitioning with separate motion estimator and half-pel refinement unit.

6 Improvements in Base model

6.1 Handshake elimination

In Sec.3, a bottleneck process was defined as the one which affects the performance of the overall model. One such process in the encoder (Fig.8.a) is the frame memory manager. Even though the computation time for frame memory manager is small compared to other

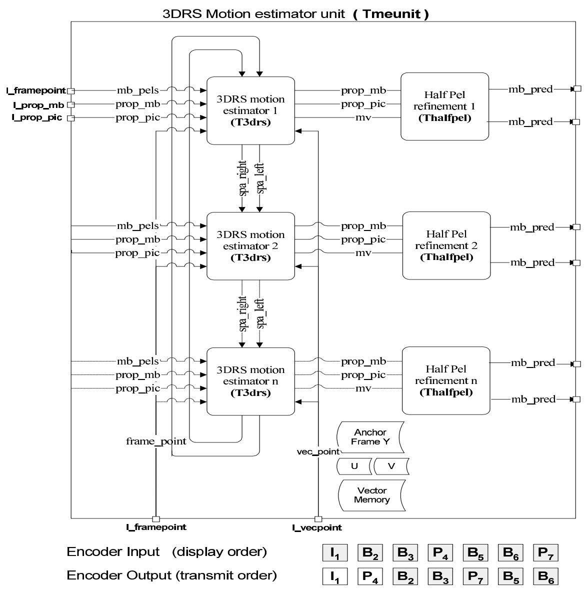
3DRS Motion estimator unit (Tmeunit)
I_framepoint
I_prop_mb
I_prop_pic
mb_pels
prop_mb
prop_pic
3DRS motion estimator 1 (T3drs)
3DRS motion estimator 2 (T3drs)
3DRS motion estimator n (T3drs)
spa_right
spa_left
mv
Half Pel refinement 1 (Thalfpel)
Half Pel refinement 2 (Thalfpel)
Half Pel refinement n (Thalfpel)
mb_pred
frame_point
vec_point
Anchor Frame Y
U
V
Vector Memory
I_framepoint
I_vecpoint
Encoder Input (display order)
I1 B2 B3 P4 B5 B6 P7
Encoder Output (transmit order)
I1 P4 B2 B3 P7 B5 B6

Figure 7: 3DRS Motion estimator unit process network

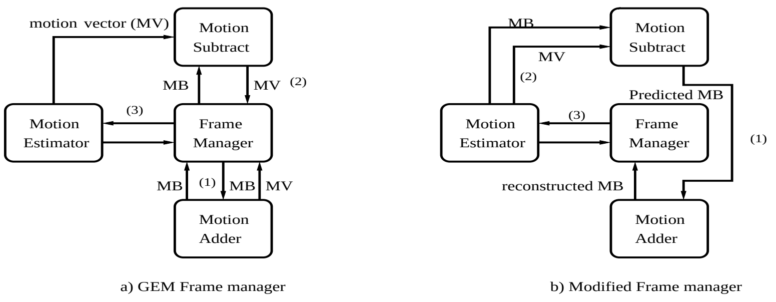

Figure 8: (a) shows the base model with handshake and (b) refined KPN model with minimum handshake

processes, it is responsible to serve requests from 3 complex processes. This bottleneck is because of the presence of centralized frame memory manager. The above KPN model was refined into a equivalent structure shown in (Fig.8.b). Here most of the handshake based implementation has been eliminated resulting in a more unidirectional dataflow model. This transformation is based on the functionality of each KPN processes and is not an automatic transformation.

6.2 Critical Path for B and P picture

This method is similar to the approach used in circuit analysis to increase the operating frequency by reducing the critical path delay. The encoding time for I/B/P pictures are different, and so the input frame sampling rate is dependent on the slowest encoding time. The critical path for B and P pictures is shown in Fig.9. Since the B picture needs

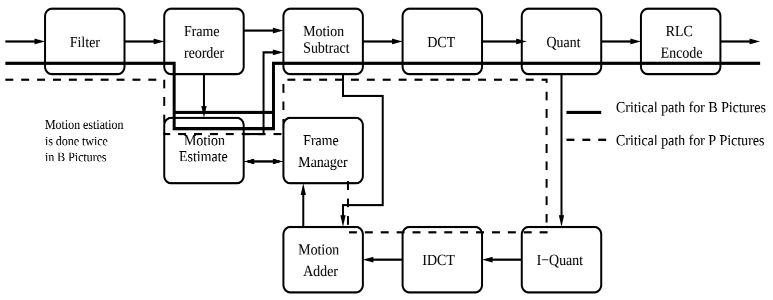

Figure 9: Critical path for B and P Picture

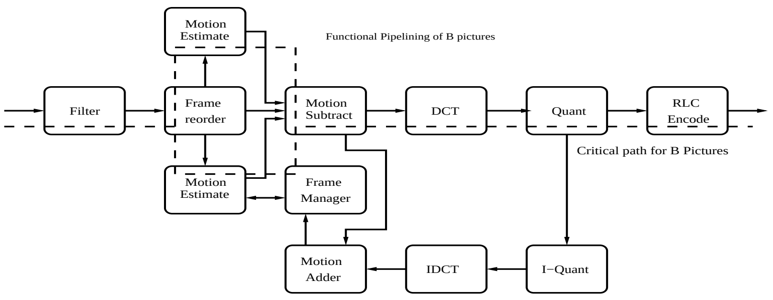

Figure 10: Modified Critical path for B picture

to do two motion estimations (with forward and backward reference I/P frames), the critical delay for B pictures was almost two times the critical delay for P picture. If we use an input frame sampling period smaller than the critical delay for B picture, the size of the input FIFO would increase for each B frame applied at the input. We need to speed up encoding time for B frame, to get almost same critical delay as P frame.

The method which has been adopted to reduce this difference is to use two motion estimators (Fig.6.2), one using forward reference frame(I/P) and other using backward reference frame(I/P). Hence by using two motion estimators(MEr) the overall critical path delay could be reduced by one MEr delay. This parallelisation is specific to B picture and the second MEr is not used in P encoding. The final simulation result (Fig.12) shows that output delay for all P/B picture was nearly constant by this scheme. This type of parallelisation where the same MB is applied to two motion estimator, which does search in parallel for same input data but with different reference frame is a kind *functional parallelism.*

7 Refined MPEG-2 Encoder Model

Based on the step by step approach suggested in earlier sections, a scalable parallel model was derived as shown in Fig.11. The 3DRS motion estimation process unit is shown in Fig.7. The basic atomic unit of scalability for motion estimation is a slice. Because of the recursive dependency between MBs of a single slice, no parallelisation is possible within a slice. Scalabiility could be extended to frame level (B frames encoding in parallel) and GOP level (two GOP encoding in parallel). At present we have done slice level parallelisation only. The main features of our model are 1) a unidirectional flow of data across the model 2) parallel motion estimation for B-pictures with respect to each reference frame and 3) partially scalable 3DRS motion estimator. The MPEG-2 encoder model shown does not implement adaptive quantisation techniques and complies only with main profile of MPEG-2 standard[11].

8 Experimental results

The refined encoder model(RM) with scalable 3DRS motion estimator was modeled as KPN using YAPI/C++ library[5] and verified to generate a bit-true output stream when compared to the existing base encoder model. Once the model was verified at functional level, it was mapped on the simulation model of CAKE multiprocessor architecture[4]. The basic unit of CAKE is a *tile*. Each tile is a symmetric multiprocessor system consisting of processor, shared memory banks, hardware engines and interconnect. Within a tile processors communicate using shared memory and MSI snoop based protocol is implemented for cache coherency[17]. Many such identical tiles were interconnected by a torus network and communicates by message passing between tiles. The number of processors used for simulations were 2,4,8 and only a single tile was used in our experiments. The behavior of CAKE multiprocessor architecture was modeled using TSS (Tool for system simulation), which is a C-based hardware description language. For comparison with base model(BM), the refined model(RM) with 3 slice parallelisation was applied so that encoding of 3 MB belonging to different slice will go on in parallel. The input sequence for our experiments was the *ISO table sequence* of size 320×288 with group of picture(GOP) size=4 and prediction distance=2. Thus the sequence pattern in IBPBPBP...

The total number of cycles required to encode a particular I/P/B frame is the latency. It is clear from Fig.12 that base model latency is 3 times that of refined model, which is because of the 3 level slice parallelism. Also the latency for P and B pictures are almost same in the refined model because of parallel motion estimation for B-pictures in the refined model. In contrast, the base model shows constant latency for B/P pictures independent of the number of processors, which proves that task level parallelism alone is not enough for good performance on multiprocessors.

Communication workload (Fig.13) could be characterized by the total number of snoop request[17] put to the snoop controller by all the CPUs in a multiprocessor and the total number of snoop request serviced by the CPU caches. The refined model has been optimised for communication by vector read/write operation, local buffering to reduce semaphore operations, shared frame memory communication and reduced handshake tokens. Hence refined model shows considerably reduced communication workload compared to the base model. The Cache miss rate (Fig.14) has also reduced significantly because of optimised buffering and reduced handshake in the refined model. Handshake implementation results in increased context switching and hence destructive interference of one process on another increases cache miss rate.

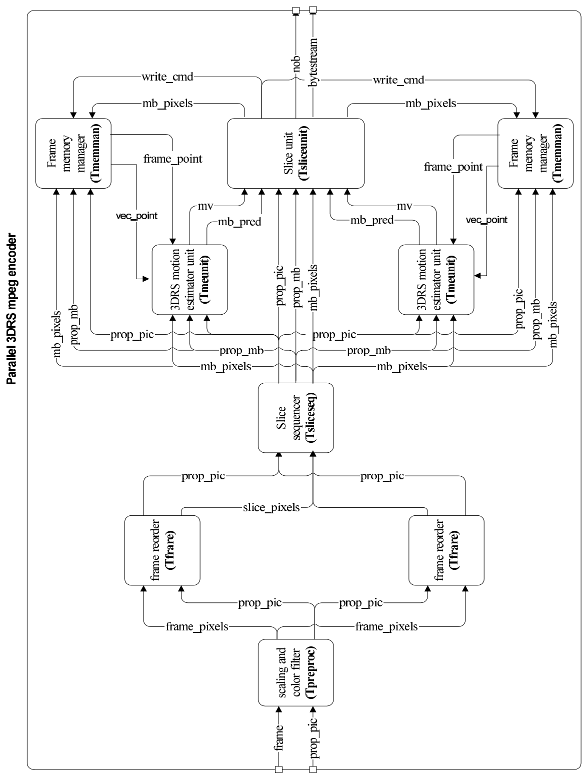
Parallel 3DRS mpeg encoder
Frame memory manager (Tmemman)
Slice unit (Tsliceunit)
3DRS motion estimator unit (Tmeunit)
Slice sequencer (Tsliceseq)
frame reorder (Tfrare)
scaling and color filter (Tpreproc)
write_cmd
mb_pixels
frame_point
vec_point
mv
mb_pred
prop_pic
prop_mb
nob
bytestream
slice_pixels
frame_pixels
frame

Figure 11: MPEG-2 Encoder Process Network

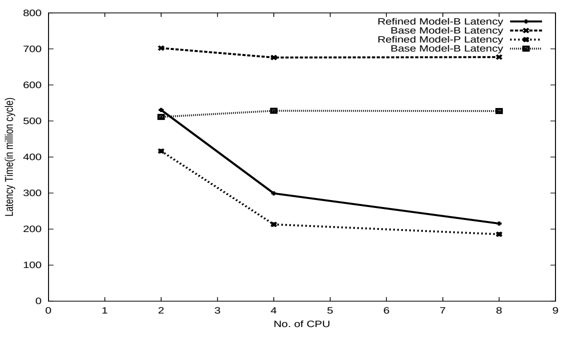

Figure 12: Latency cycles

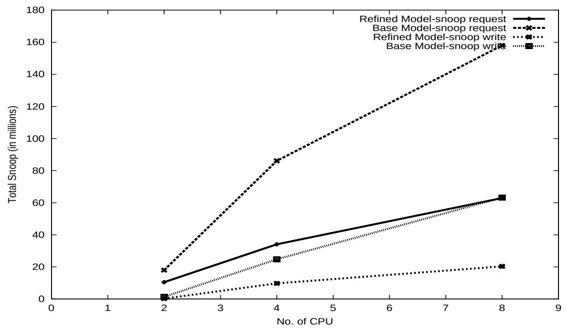

Figure 13: Total snoop request to bus

9 Conclusion

From the experimental results described the following conclusions was derived:

- The refined model show good scalability and improved execution time compared to the MPEG-2 base model (Section 4). Refined model captures all levels of parallelism present in the MPEG-2 encoding with optimised communication between process. All metrics (cache miss rate, latency, bus snoops) show improved performance in the refined model. One main result of our experiment is that task-level parallelism alone is not sufficient for good performance on multiprocessor but data and functional parallelism are also essential. Another important point is that there must be more parallelism in the program than in the designated architecture to ensure that processors are not idle. Such a property is called as parallel slackness[18].

- One main factor considered for analysis was the latency for P and B pictures which scales linearly with the amount of slice parallelism present in the new model. For the new model consisting of 3 slice level parallelism, latency of P/B pictures was reduced by a factor of 3 compared to the base model (Fig.12). Also, both P and B picture shows almost same latency because of parallel motion estimation for B pictures.

- The Data Dependency Graph(DDG) (Sec.5) provides a simple method to apply data parallelism in any KPN model to convert it into a scalable KPN model. DDG was applied to many motion estimation algorithms, the maximum scalability possible for each was studied and a scalable 3DRS motion estimation KPN model was implemented for MPEG-2 encoder.

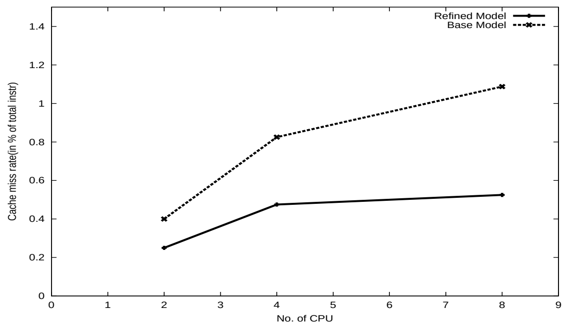

Figure 14: Cache Miss rate

10 Acknowledgement

References

[1] Akramullah.S.M, Ahmad.I, and Liou.M.L. A data-parallel approach for real-time mpeg-2 video encoding. *Journal of Parallel and Distributed Computing*, 30(2):129–146, November 1995.

[2] Ke Shen and Edward J. Delp. A spatial-temporal parallel approach for real-time MPEG video compression. In *ICPP, Vol. 2*, pages 100–107, 1996.

[3] D. Barbosa and J. Kitajima. Real-time mpeg encoding in shared-memory multiprocessors. In *International Conference On Parallel Computing Systems, 1999*, 1999.

[4] Paul Stravers and Jan Hoogerbrugge. Homogeneous multiprocessing and the future of silicon design paradigms. In *Proc. International Symposium on VLSI Technology, Systems, and Applications (2001 VLSI-TSA)*, April 2001.

[5] E.A. de Kock, G. Essink, W.J.M. Smits, P. van der Wolf, J.Y. Brunel, W.M. Kruijtzer, P. Lieverse, and K.A. Vissers. Yapi:application modeling for signal processing systems. In *Proc. 37th Design Automation Conference (DAC'00)*, pages 402–405, June 2000.

[6] Gills Kahn. The semantics of a simple language for parallel programming. In *Proc. IFIP Congress 74*. North Holland Publishing Co, 1974.

[7] Twan Basten and Jan Hoogerbrugge. Efficient execution of process networks. In *Communication Process Architectures*. IOS Press, 2001.

[8] H.M.Jong, L.G.Chen, and T.D.Chiueh. Parallel architectures for 3-step hierarchical search block-matching algorithm. *IEEE Transactions on Circuits and Systems for Video technology*, 4(4):407–416, 1994.

[9] Chen.L.G, Chen.W.T, Jehng.Y.N, and Chiueh.T.D. A effective parallel motion estimation algorithm for digital image processing. *IEEE Trans. on Circuits and Systems for Video Technology*, 1(4):378–385, December 1991.

[10] Basant Kr. Dwivedi, Jan Hoogerbrugge, Paul Stravers, and Balakrishnan. Exploring design space of parallel realizations: Mpeg-2 decoder case study. In *Proc. of CODES 2001*, pages 92–97, April 2001.

[11] *Information technology - Generic coding of moving pictures and associated audio information: Video.* ISO/IEC 13818-2, 1996.

[12] Atul Puri and R.Aravind. Motion compensated video coding with adaptive perceptual quantisation. *IEEE Trans. on Circuits and Syst. for Video Tech.*, 1(4):351–61, December 1991.

[13] Joan L. Mitchell William B. Pennebaker, Chad E. Fogg and Didier J. LeGall. *MPEG VIDEO COMPRESSION STANDARD.* Chapman & Hall, New York, 1996.

[14] Martijn Rutten, O.P.Gangwal, Jos.V.Eijndhoven, E.Jaspers, and Evert-Jan Pol. Hardware/software codesign of reusable mpeg coprocessors for eclipse(submitted). *Intl. Conf. on Computer Design(ICCD'02)*, September 2002.

[15] G.de Haan, W.A.C.Biezen, Henk Huijgen, and O.A.Ojo. True-motion estimation with 3d-recursive search block matching. *IEEE T. on Circuits and Systems for Video Tech*, 3(5):368–379, OCTOBER 1993.

[16] G.D.Micheli. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, 1994.

[17] Culler.D, Singh.J.P, and Anoop Gupta. *Parallel Computer Architecture: A Hardware/Software Approach.* Morgan Kaufmann, 1998.

[18] Skillcorn.D.B and Talia.D. Models and languages for parallel computation. *ACM Computing Surveys*, 30(2):123, June 1998.