New Approach to Architectural Synthesis: Incorporating QoS Constraint

Harsh Dhand
Philips Research
Bangalore
harsh.dhand@philips.com

Basant Dwivedi
Calypto Design Systems (I)
Pvt. Ltd.,
Noida
basant@calypto.com

M Balakrishnan
IIT Delhi
New Delhi
mbala@cse.iitd.ernet.in

ABSTRACT

Embedded applications like video decoding, video streaming and those in the network domain, typically have a Quality of Service (QoS) requirement which needs to be met. Apart from being a design constraint, it can also be considered as a flexibility that the design does not have to work under worst case data condition. For example, in the case of video decoding with variable decoding time for individual frames, it may be adequate that only a fraction of frames (say 90%) needs to be decoded. In this work, we propose a novel method of exploiting this flexibility for efficient partitioning and mapping in the architectural synthesis of the application at hand. We translate QoS specification of the overall application to the time constraint on the individual components constituting the application and use this knowledge in an optimum synthesis technique based on Mixed Integer Linear Programming (MILP). We study this in the context of MPEG2 Decoder and show that the approach can be used to obtain optimal time of execution as well as energy reduction while meeting the QoS requirements.

Categories and Subject Descriptors

C C.3 [**Computer Systems Organization**]: Special Purpose and Application-Based System—*Real-time and embedded systems*

General Terms

Design, Measurement, Performance

Keywords

Quality of Service, Soft Real Time Constraints, Mapping, Partitioning, Process Network

1. INTRODUCTION

Using quality of service in video decoding applications has been a key area of research in Networks [1] and advanced media applications [4]. Steinmetz et al. [5] have given both qualitative and quantitative description of QoS for Multimedia systems. They have enumerated various parameters which can be used for QoS description like Perception QoS(tolerable synchronization drift, visual perception), System QoS(CPU rate, memory usage), Communication QoS(Packet size, bandwidth) etc.

The importance of considering QoS requirements in the architectural synthesis, is also illustrated by Marculescu et al. [6, 7]. In these works, authors have shown that incorporation of QoS constraints allows one to achieve performance which is sometimes as much as an order of magnitude better in relation to considering worst case behavior. The ability to explore several design alternatives while trying to satisfy QoS requirements is of crucial importance, and their work attempts the same. Work on incorporating QoS in synthesis is described in detail in section 2.

Multiprocessor System-on-Chip(MpSoC) Synthesis for embedded media applications can be done in a variety of ways such as scheduling [8], process network mapping [9] etc. For this synthesis, the end-to-end deadline on the complete application, be it hard or soft, must be distributed over component tasks [10]. Further, partitioning and mapping of the application onto multiprocessor architecture during MpSoC synthesis, requires the knowledge of how the various components of the application perform on processors and what are their memory requirements. Details about the architectural synthesis methodology that we have chosen are given in section 6. Instead of using the worst case computation times of processes on various processors being explored, QoS specifications allow us to allocate much smaller times. This is because it is not required to successfully complete 100% of the process in time. This results in a more relaxed mapping constraints and thus a smaller and less power consuming hardware.

The objective of our work is to integrate the task of designing multiprocessor architectures with meeting QoS requirements in mapping. This is achieved by computing a suitable requirement of QoS for the constituent processes of the application. However while the mapping is to be done for individual processes, the QoS is specified for the overall application. Hence, there is a need for finding QoS constraints that can be used for the individual processes. In this work, we address the problem of distribution of QoS constraints over individual application components(processes) and propose a methodology for the same. We further show the application of our approach with the help of MPEG2 video decoder application. In this case study, we take the system QoS, the processor speed required, as the primary parameter which would influence the perception QoS, represented by frames decoded per second.

The rest of the paper is organized as follows. Section 2 describes in detail the importance of QoS in architectural synthesis, with illustrating examples from literature. Section 3 gives the statistical interpretation of incorporating QoS, including the qualitative

EMSOFT'06, October 22–25, 2006, Seoul, Korea.

results from application of statistical techniques for incorporating QoS in synthesis. Section 4 describes the mapping and partitioning phases of architectural synthesis for an application. Section 5 describes the application and using cumulative distribution functions in analysis. The algorithm developed for optimal QoS distribution is described in Section 6. The experimental setup is described in detail in section 7. Results obtained are enumerated in Section 8. Discussion on using QoS distribution for energy minimization is also in the same section. Section 9 concludes and identifies some of the interesting applications of using QoS in synthesis.

2. QOS FLEXIBILITY IN SYNTHESIS

Designing based on the worst-case execution time model guarantees that no timing requirement is broken. However, for large classes of applications, the soft real-time systems, violating a timing requirement, though not desirable, is tolerated provided that this happens with a sufficiently low probability. Whereas in the case of critical systems, the designers stress meeting deadlines at the expense of product cost, in the case of soft real-time systems, cost reduction is a strong incentive for using cheap architectures.

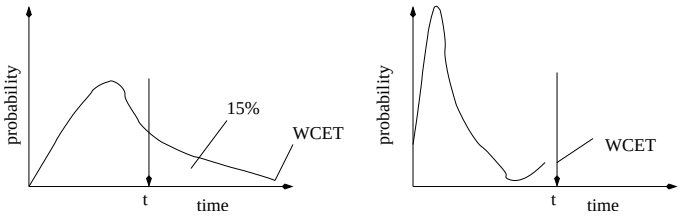


Figure 1: Requirement of processors for WCET

Sorin et al [13] have illustrated the difference between following a Worst Case Execution Time Approach(WCET) on one hand, and using Execution times and QoS requirements on the other, for finding right processor for a process. Let us consider two processors and a task which runs on them. The probability density function of the task execution time (ETPDF) on the processor 1 (inexpensive and less computation power) is depicted in the left subfigure of Figure 1. If the imposed deadline of the task is 't' as shown in the figure, then the processor cannot guarantee that the task will always meet its deadline, as the WCET of the task exceeds the deadline. If no deadline misses were tolerated, a faster and more expensive processor (P2) would be needed. The ETPDF of the task on the faster processor is depicted on the right in Figure 1. In this case, the more expensive processor guarantees that no deadlines are missed. However, if a miss deadline ratio of at most 15% is tolerated, then even the processor 1 would suffice.

A problem that has been addressed formerly is finding the percentage cases in which deadlines miss, given a multiprocessor platform and an application. This is achieved through a large number of simulation runs. The problem when stated in the reverse manner is the more interesting problem of synthesis and is objective of this work: " given a multiprocessor hardware architecture and an application expressed as communicating processes find a process mapping such that the application runs on the architecture meeting the desired constraints, and that the architecture cost is minimized."

One way to address the problem is to use statistical data obtained from simulation runs e.g. the worst case and average execution times and higher moments of the distribution of execution times.

2.1 Illustrating Examples

[13] have given sample cases to show that only a complete analysis of the simulation data would result in correct solution to the incorporation of QoS into synthesis. Simply using the average behaviour does not help as shown in the following example.

Let the application be as shown in Figure 2.2a. The circles denote the tasks, their shades denote the processors they are mapped onto. The solid disks show the inter-processor communication. The arrows between the tasks indicate their data dependencies. All the tasks have period 20 and the deadline of the task graph is 18. Tasks A, B, C, and D have constant execution times of 1, 6, 7, and 8 respectively. Task E has a variable execution time whose probability is uniformly distributed between 0 and 12. Hence, the average (expected) execution time of E is 6.

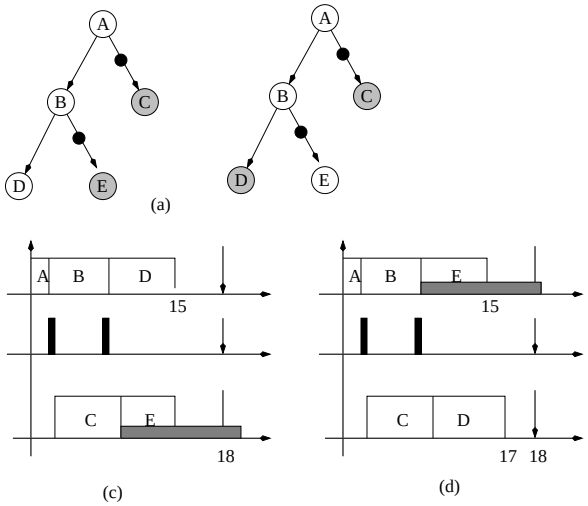


Figure 2: Effect of mapping on deadline miss

The inter-processor communication takes 1 time unit per message. Let us consider the two mapping alternatives depicted in Figure 2a and 2b. The two Gantt diagrams in Figure 2c and 2d depict the execution scenarios corresponding to the two considered mappings if the execution of task E took the expected amount of time, that is 6. The shaded rectangles depict the probabilistic execution of E. A mapping strategy based on the average execution times would select the mapping in Figure 2a as it leads to a shorter response time (15 compared to 17). However, in this case, the worst case execution time of the task graph is 21. The deadline miss ratio of the task graph is 3/12 = 25%. If the stochastic nature of the execution time of task E is taken into consideration, the second mapping alternative should be chosen, because of the better deadline miss ratio of 1/12=8.33%. If, however, the worst case response times are considered instead of average ones, then the second mapping alternative will be chosen.

Approaches based on worst case execution times can be dismissed by means of very simple counter-examples.

Consider a process A which can be mapped on processor P1 or on processor P2. P1 is a fast processor with a very deep pipeline. Because of its pipeline depth, mispredictions of target addresses of conditional jumps, though rare, are severely penalized. If A is

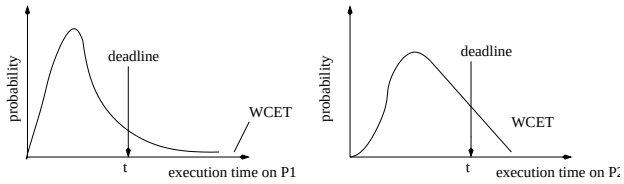


Figure 3: Effect of processor on deadline miss

mapped on P1, its ETPDF is shown in Figure 3a. The long and flat density tail corresponds to the rare but expensive jump target address misprediction. If is mapped on processor P2, its ETPDF is shown in Figure 3b. Processor P2 is slower with a shorter pipeline. The WCET of task A on processor P2 is smaller than the WCET if A ran on processor P1. Therefore, a design space exploration tool based on the WCET would map task on P2. However, as Figure 3 shows, the deadline miss ratio in this case is larger than if task A was mapped on processor P1.

2.2 QoS in our framework

There is a requirement for incorporating QoS constraint in synthesis. The synthesis process involves selection of computation modules, memory modules, communication architecture and mapping of processes of application onto compute units and channels on memory modules. Partitioning and mapping of application onto multiprocessor environment, requires the knowledge of how the various processes of the application perform on processors and what are their memory requirements. Instead of using the worst case computation times of processes on various processors being explored, QoS specifications allow us to allocate much smaller times. This is because it is not required to successfully complete 100% of the process in time. This results in a more relaxed mapping constraint and thus a smaller and less power consuming hardware.

While exploring the mapping of processes of MPEG2 decoder onto processors, the worst case execution times of the processes of MPEG2 are fed to the framework. This does not give the best possible solution. That is why work was done towards deriving a methodology based on simulation statistics and QoS specification, for obtaining a better DSE. It is my contention that working with QoS constraint is likely to produce a much better solution.

3. STATISTICAL INCORPORATION OF QOS

In statistics, various inequalities are used to express the behavior at the tail of PDFs. Whether they can be used for expressing QoS and are valid for our applications is discussed here.

Let α denote the soft real time constraint. α is related to QoS as 1 - QoS(expressed as a fraction). So, α is the maximum fraction of cases, in which the application is allowed to miss the performance deadline. For statistical distribution, we know the Chebyschev's inequality which states that

$$Pr[|X-c| \geq \varepsilon] \leq 1/\varepsilon^2 E(X-c)^2 \quad (1)$$

Using mean and standard deviation in the above equation, we get:

$$Pr[|X-\mu| \geq \varepsilon] \leq VarX/\varepsilon^2 \quad (2)$$

Using this and the definition of α we get,

$$Pr[|X-\mu| \geq \varepsilon] \leq VarX/\varepsilon^2 \leq \alpha \quad (3)$$

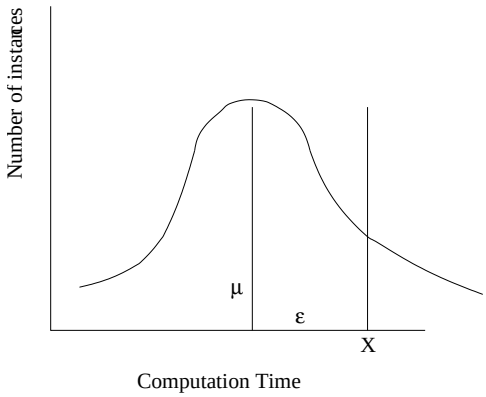


Figure 4: Chebyschev's Inequality for SRT constraint

i.e for the worst case, $\varepsilon = (VarX/\alpha)^{1/2}$

$$Hence\, X = \mu + (VarX/\alpha)^{1/2} \quad (4)$$

The idea of relating the mean and the standard deviation is illustrated in Figure 4. Given the mean, Standard Deviation, and the SRT constraint α, we have X, which can be used for the mapping process.

- Chebyschev's inequality assumes independence of measured variables. Processes in a multimedia application typcially depend on external data and possibly having a correlation amongst themselves.
- Also chebyschev's inequality gives loose bound. This is actually what was seen from the experiments that we did on the computation time data for MPEG2 processes.

Thus we tried inequalities with tighter bounds and which support partial dependence. One such bound is the Chernoff-Hoeffding bound. Chernoff-Hoeffding bounds are fundamental tools for the tail probabilities, i.e. the probabilities of deviation from mean, for bounded and independent random variables. [3] have given a more generalized form of the inequality and shown that it is applicable for variables with limited independence.

The tail inequalities could be of the following form:

Additive form:

$$Pr[X > \lambda] < fn()\ Or\ Pr[X > \lambda + \mu] < fn() \quad (5)$$

Multiplicative form:

$$Pr[X > (1+\delta)] < fn(). \quad (6)$$

The function on the right side of the above equations could be a function of n(the number of data values), α or δ, and μ, the mean. While Chebyschev's inequality was expressed in additive form, Chernoff-Hoeffding is in multiplicative form.

[3] have shown that the inequality can be applied to variables for limited independence and have given the following forms of inequalities.

$$Pr[X \geq (1+\delta)\mu] \leq U(n,\delta,\mu) \leq F(n,\delta,\mu) \leq G(n,\delta,\mu) \quad (7)$$

In each case put $f(\delta) = \alpha$ and solve for δ, and substitute to obtain X. Tighter the bound the more complex solving for δ or ε.

For example in the above forms,

$$U(n,\delta,\mu) = (({}^{n}C_{i*})(\mu/n)i*)/^{\mu(i+\delta)}C_{i*} \; where \; i* = \mu\delta/(1-\mu/n) \tag{8}$$

The easiest form here is that of function $G(n,\delta,\mu)$

$$G(\delta,\mu) \leq e^{(\delta^2)\mu/3} \leq \alpha \tag{9}$$

Gives: $X = (1+3ln(1/\alpha)/\mu)\mu$

The above inequalities actually gave too tight bounds. But even if the result of the above inequalities cannot be used directly, it gives us a means of expressing the value to be used as a function of mean and standard deviation. Before describing the approach that we followed for distributing the QoS, we describe in detail Architectural Synthesis, specifically the MILP approach that we used for it.

4. QoS IN ARCHITECTURAL SYNTHESIS

QoS constrained applications like video decoding are required to give a certain throughput. Having a QoS constraint, as opposed to having a hard real time constraint implies that though the application has a constraint of 1/30 seconds a frame, but it is acceptable if it does not complete each time and still meets the required viewing constraints. As discussed in [6], considering the above flexibility offered by soft-real time constrained applications in the form of QoS, results in a cheaper architectural solutions. We discuss next, what MpSoC synthesis involves and where QoS constraints can be incorporated during MpSoC synthesis.

We consider applications represented as process networks (PN) [11, 12]. However, our approach is not limited to process networks and it can be applied to any other periodic task graph as well. Given an application in the form of a process network and a component library, synthesis of the architecture consists of the following to minimize total cost or energy.

1. Allocation of processors, local and shared memories
2. Binding of processes to processors and queues to local or shared memories
3. Allocation of communication components such as buses

Figure 5 shows an instance of a synthesized architecture. In this example, the application process network is composed of 3 processes and 3 queues. The synthesized architecture consists of 2 processors, 1 local memory and 1 shared memory. *Queue 1* is mapped to the local memory of *Processor 1* because reader and writer processes of *Queue 1* are mapped here. On the other hand *Queue 2* and *Queue 3* are mapped to the shared memory as their reader and writer processes are mapped to two different processors.

When the *objective function* of the synthesis problem is to minimize the hardware cost, then the total cost of processors, local memory modules, shared memory modules and interconnections cost is minimized. On the other hand, in case of energy minimization, one tries to minimize energy consumed during execution of processes, memory accesses and data transfer on the interconnection network.

A MpSoC synthesis framework for process network such as the one described in [14] using (Mixed Integer Linear Programming - MILP) or described in [9] using a heuristic based framework can be used to solve the above synthesis problem. The input is the PN description of the application, and an architectural component library which contains compute units, memory modules and interconnection network components. Alongwith each compute unit, there is a

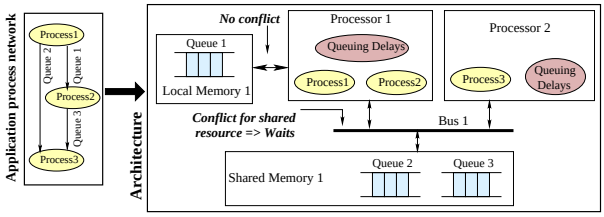


Figure 5: MpSoC Synthesis example

local memory. There could also be a number of shared memories which are connected to the compute units through interconnection network as shown in figure 5. Apart from these, in [14, 9], authors have also assumed that performance requirements are specified for individual processes of the process network.

A compute unit offers number of time units equal to its clock frequency (cycles per second). This must accommodate computation overheads of processes mapped, context switch overheads and interconnection network overheads. Let tp_{ik} be the binary variable denoting mapping of process P_i onto processor PR_k. ncy_{ik} is the number of cycles taken by the process P_i when mapped onto processor PR_k for one iteration. $iter_i$ is the real time constraint on process PR_i indicating the number of iterations per second to be performed by the process P_i. If the frequency of the processor PR_k is $freq_k$, then the *Overall Performance Constraint* is expressed as follows.

$$\begin{array}{c} \forall k: \quad \sum_i tp_{ik} \times ncy_{ik} \times iter_i + \\ + other\_overheads \leq freq_k \end{array} \tag{10}$$

The objective function to minimize hardware cost is follows.

$$\begin{array}{c} minimize: \quad compute\_units\_costs \; + \; Memory\_costs + \\ interconnection\_network\_cost \end{array} \tag{11}$$

Let us assume that the QoS is specified in terms of fraction (say 90%). So, if instead of using worst case behavior, we incorporate QoS in the performance constraints, it would reduce ncy_{ik}. In Equation 10, we note that reducing ncy_{ik} will give processor PR_k more slack. In effect, either $freq_k$ can be reduced resulting in low energy execution or more processes can be mapped onto PR_k possibly reducing hardware cost.

With the help of MPEG2 decoder application, in the next Section, we describe how an application needs to be analyzed so as to generate enough information for QoS distribution across the whole application.

5. APPLICATION ANALYSIS

The methodology for finding the QoS described is applicable for any application expressed as communicating tasks. Typical application descriptions, like PN models, are used for architectural synthesis, hence the methodology of QoS distribution proposed has been applied to PN model. To know the individual QoS constraints, and the methodology for finding it, the PN model of MPEG2 decoder (figure 6) was experimented with extensively.

As will be explained in section 6, we require the total computation time of various processes constituting the application. This can be obtained by either simulation or estimation. Presently, the computation time is measured by running the application on a multiprocessor(MP) VLIW simulator [2], which was instrumented by us to give the profile of independent threads (processes). The architecture consists of VLIW processors, with four issue slots, local instruction and data caches, a shared memory. The system is

composed by assembling various processing elements, memories, peripheral devices and interconnects. The MP simulator provides a component library of parameterized and cycle accurate simulation models of memories, VLIW processor, shared bus and non-intrusive thread-aware function-level profiler. The final result of the analysis is the QoS distribution, using the computation times and input QoS. The accuracy of the methodology developed depends only on the fidelity of the simulator or estimator in producing consistent results.

For describing the application behavior and the meaning of QoS, we have adopted the same convention as done by Manolache et. al. [13]. Their focus is on task priority assignment and scheduling of tasks, whereas with the input application model we have taken, we only have to incorporate the QoS into mapping algorithm.

5.1 Application characteristics

5.2 Sample application

We have taken MPEG2 decoder process network as our example application. Modeling the application as Process network solves the scheduling problem. Mapping this PN model onto a multiprocessor architecture needs to be performed. The mapping of the application has been done by taking the throughput constraints into consideration, hence a run time scheduler can now execute the application on the architecture and the solution meets the deadlines. We aim to still meet the constraints provided by QoS. The algorithm suggested brings a change in the mapping solution, it is not at the run-time scheduling level.

The process network (PN) of MPEG2 decoder consists of 6 processes: Header Decoding (P_1), Slice Decoding (P_2), Inverse Discrete Cosine Transform (P_3), Motion Prediction (P_4), Addition(P_5) and Process Store(P_6). This is shown in figure 6, with the arrows between processes indicating the data flow between the processes.

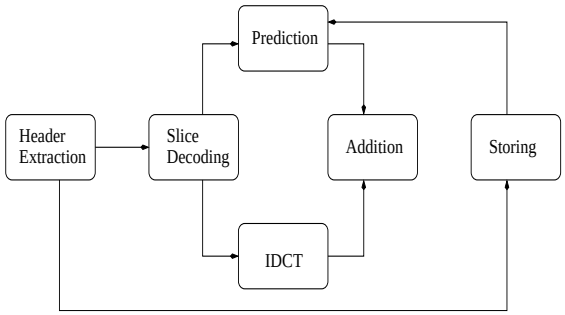


Figure 6: MPEG2 Decoder PN Model

We generated various video streams, encoded such that they contain only I and P frames. The methodology developed can be used for more generic input consisting of B frames also. Analyzing various sample videostreams, we found that a videostream typically consists of one I frame followed by 10 or so P frames. The decoding times of I frames and P frames were obtained from simulation and their mean values are as shown in table 1. P_1 to P_6 are the processes in MPEG2 Decoder model as explained in Section 2.2.

	P_1	P_2	P_3	P_4	P_5	P_6
I Frames	514	6332	2989	194	3205	2020
P Frames	143	5370	2681	2586	3653	1846

Table 1: Mean Computation Time (in 1000 cycles)

As expected, for processes P_1(Header Decoding), P_2(Slice Decoding) and P_3(IDCT) the computation times for I frames are higher, while for P_4(Prediction) it is mainly the P frames where computation is high. For P_5(Addition), P frames take slightly higher time owing to the input from prediction and IDCT. For P_6(Storing), I frames have greater computation time. Owing to these variations, different data sets were taken for I Frames and P frames.

Four hundred such computation time values each for I and P frames, from 8 streams (Common Intermediate Format (CIF) Picture size: 352x288, 30 frames/sec) were taken to perform the analysis. The process of generating one to n frame streams, running the simulations and obtaining these values has been fully automated. Thus, given sample video streams and the number of sample data values to be obtained, the simulation proceeds automatically to give these data points. Though simulation of MPEG decoding typically takes a large time, this task of measuring the computation times is a one time process, a pre-processing step to the architecture exploration phase. If a good estimation technique is used to obtain these computation times, then the this data collection which has presently taken one day, could be done in a couple of hours. Moreover, for the set of use-cases taken as input, the QoS distribution obtained, is the optimal distribution that can be used for architectural synthesis.

We used **Cumulative distribution function, CDFs** to find the appropriate values of QoS constraints for processes and also the respective bounds to be considered. Let us denote by X the random variable associated with the number of clock cycles used by a task instance. We will use the cumulative density of probability function, CDF, associated with the variable X,. This function reflects the probability that a task instance finishes before a certain number of clock cycles.

$CDF(x) = P(X \leq x)$

Consider Fig 3(b), the CDF of process 6, shows that 80% of considered sample cases have a value below 1.85e+06, while 99% have it below 1.855e+06. A QoS of 80% for process 6 corresponds to the fact that if given a computation time X for execution, atleast 80% of the cases P6 will execute correctly. This establishes the direct relation between the values obtained by CDF and QoS required. CDFs are obtained by finding the computation times from a large number of simulation runs and then sorting it in the ascending order. Thus for a given QoS for a process, for a value on the y axis, the corresponding computation time corresponds to the value on the x axis.

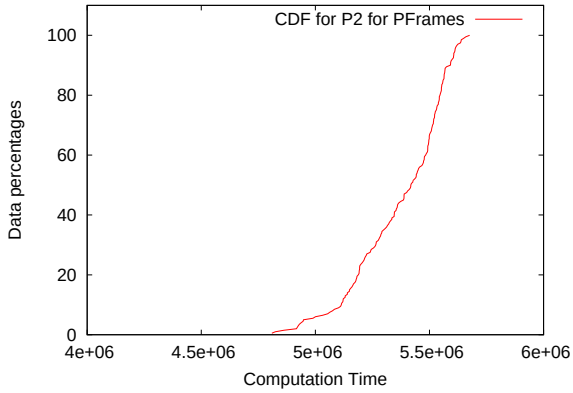
Please note that the computation times of the processes are input data dependent. The CDFs of various processes differ. From the total time of computation, CDF of the overall application can also be plotted. Fig 7(c) shows the overall CDF for P frames.

5.3 Optimal QoS for Processes

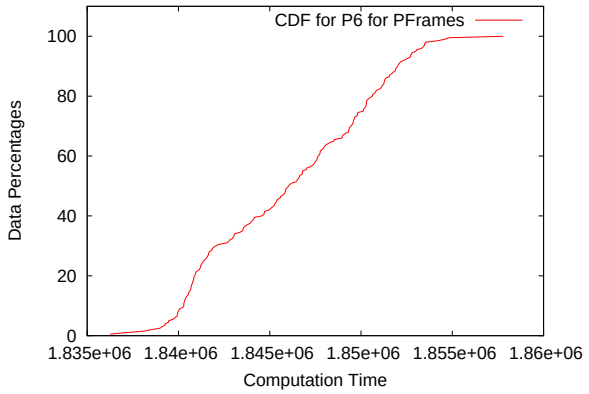
Typically the overall QoS for an application is given. A variety of QoS for individual tasks can be chosen, all of which achieve the same overall application QoS. However, as we show in this and the next section, our algorithm achieves am optimal QoS distribution. The distribution is optimal because the chosen set results in a achieving the least cost architecture for the overall application. Let the QoS for process P_i is denoted by S_i and the QoS for the overall application by S. For simplicity of equations, S is taken as a fraction rather than %. The computation time to be allotted to process P_i is denoted by T_i, while for the overall application it is T. Clearly,

$$\sum_i T_i = T \tag{12}$$

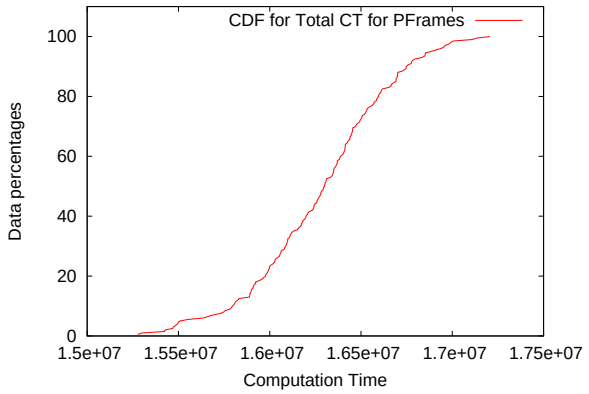
But no such simple equation exists for relating process QoS(S_i) to application QoS(S). The objective of our work now reduces to



(a) CDF for P_2



(b) CDF for P_6



(c) CDF for Total CT

Figure 7: Cumulative Distribution Function for P Frames (Computation Times in cycles)

finding an optimal S_i for each P_i, such that we can minimize T, while still meeting the overall QoS requirement of S. Note that T_i and S_i for each process are related by the corresponding CDF, as shown in figure 7(a) and 7(b) for P_2 and P_6. If all the processes were independent, from probability theory, it is easy to see that

$$\prod_i S_i = S \tag{13}$$

This would lead to a result that for achieving 90% QoS (i.e. S=0.9) for the application, each process would require a QoS of $(0.9)^{1/6}$, which is very close to to 1. But this is almost never true as the computation time of all the processes depends on the nature of input data in a similar manner, i.e. there is "significant" correlation between computation times of different processes. Let us define $CT(P_i, d_k)$ as the computation time for process P_i for data d_k. The data in case of video decoding is a video frame. Let us consider two data points d_1 and d_2. Two processes P_i and P_j are directly correlated as follows.

$$CT(P_i,d_1) > CT(P_i,d_2) \Rightarrow CT(P_j,d_1) > CT(P_j,d_2) \tag{14}$$

In case of inverse correlation between P_i and P_j, with increase in CT of P_i, there is decrease in the computation time of P_j. Both direct and inverse correlations exist and help us in achieving better overall QoS than the worst case. In case of direct correlation between P_1 and P_2, if the time allocated to both T_1 and T_2 are reduced from the worst case, the QoS does not fall as a product. This is because for most data where P_1 misses its deadline, P_2 also misses its deadline. For inverse correlation between P_1 and P_2, one can increase T_1 and reduce T_2 to achieve the same overall QoS. If the objective is to minimize T, given an S_{req} for the overall application, how does it relate to the time that can be read from the complete application CDF corresponding to S_{req} (figure 7(c))? This time represents the lower bound on T given S_{req}, and we denote it by T_{lb}

$$T_{lb} = value\ of\ T\ at\ S_{req} \tag{15}$$

in the CDF of T. It is clear that for achieving a specified application QoS S_{req}, we need this minimum time, but do not know how to distribute this time over the individual processes effectively. One upper bound on Time T can be found by assuming the processes to be independent and reading the time for each process CDF corresponding to $S_i = \sqrt[m]{S_{req}}$ for a process graph with m processes. This corresponds to almost allocating worst case time for each process. In the next section we address the problem of finding an "optimal" close to the lower bound.

$$T_{ub} = \sum T_i \tag{16}$$

where T_i corresponds to S_i, S_i being worst case, defined above.

6. OBTAINING OPTIMAL PROCESS QoS

As explained in section 5.2, correlation amongst processes results in better overall QoS for given individual QoS than the expected worst case. Hence if we actually measure the overall QoS for various individual QoS, then we will be able to find just the right individual QoS that would suffice to obtain a desired overall QoS. This is the key idea we have used to find optimal process QoS and is described by algorithm 1. In the algorithm, α denotes assumed QoS for the processes. The algorithm has two distinct stages. In the first stage (Steps 1 to 6), we obtain a plot between S and α where all S_i have the same value α. In the second stage (Steps 7 to 10) we vary the individual S_i values to minimize the total time T, while maintaining the overall QoS to S_{req}.

In step 2 of the algorithm, we take up different values of α all equated to S_i (ranging between 90% and 99%) and find corresponding value of S from the computation time data. A failure in any one

Algorithm 1 Finding Optimal QoS Distribution

Input: Computation Time Data (Various CDFs, data from simulation or estimation) and S_{req}
Output: Optimal $\{S_i\}$

1: **for all** α = min to max **do**
2: Let S_i be α for all i
3: Compute corresponding S
4: **end for**
5: Plot S against α
6: Read α' corresponding to S_{req} and initialize $S_i = \alpha'$ for all i
7: **while** one of steps 8 or 9 is feasible **do**
8: Select and decrease one S_i while maintaining $S \geq S_{req}$
9: Choose a pair of processes P_i and P_j and reduce S_i and increase S_j such that overall time T is reduced, maintaining $S \geq S_{req}$
10: **end while**
11: Output $\{S_i\}$

process for a data point is taken as a failure to meet deadline for the complete application for that data. Hence, from the data set, we identify the cases which correspond to failure to meet one or more individual S_i, corresponding to each time T_i obtained from α. 100− (% failures) = S by definition. Thus we can find S for different S_i. Note that if S_i are each 98%, the worst case S possible is 88% , since the violations caused by each might just add up as expected from $\prod_i S_i = S_t$ for independent processes. Figure 8 shows a plot of the assumed individual QoS for each process and the total application QoS observed. The violation cases actually overlap, resulting in a better QoS. As shown in figure 8, the QoS achieved is much better than the worst case expected. Also, the worst case QoS, decreases faster than the obtained QoS, with decrease in S_i.

Once a plot as shown in 8 is obtained, **Reverse Mapping** is used in Step 6. **Given S_{req}, this plot is used to find S_i of each process, such that the overall constraint is met.** e.g. given an overall constraint of 90%, we know that a constraint of 96% each would achieve it (instead of worst case of 98.4%). From each of these S_i values corresponding T_i and thus T can be computed.

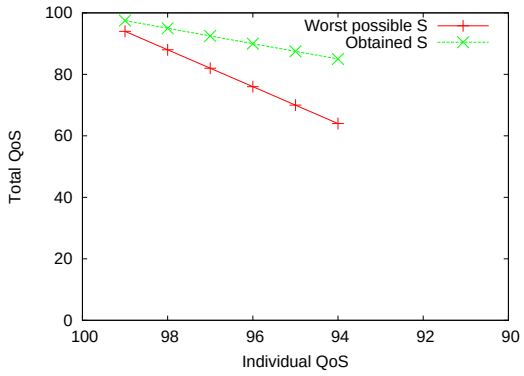


Figure 8: Effect of Individual QoS on overall QoS, QoS expressed in %

We can actually see correlation in the failure cases of individual processes. P_1, P_2 and P_3 typically fail together. P_4, P_5 and P_6 fail together, though not as consistently. That gives us some idea of the correlation amongst processes. P_4, P_5 and P_6 are higher for one type of data, while P_1, P_2 and P_3 are higher for another type.

Note that in the solution obtained from stage 1 of the algorithm, all the processes are given the same S_i. Having same value for each S_i would not give us the best possible set $\{S_i\}$. While obtaining this initial solution, we have used only positive correlation among processes to our advantage. Steps 7 to 10 in the algorithm are ways to improve the solution in terms of the total computation time of the application. The steps involve exhaustive search for solution $\{S_i\}$, with the objective being to reduce T. First is simply to lower S_i for each i. Clearly T is reduced. At each step we check that S_{req} is not violated. This is done in the same way as originally the value of S was seen, i.e. how many cases are violated. In our case, only lowering of S_2 was possible.

Next we consider the processes in pairs, increase the S_i of one to make space for decrease in S_j of another such that overall T is reduced. For this step, a way of choosing the pair of processes can be considered. Take P_i to be such that increase in S_i gives least increase in T while let P_j be such that decrease in S_j results in maximum decrease in T. This can be easily seen from the gradient of CDFs of the processes. Then increase in S_i and decrease in S_j improves our solution.

Both the steps 8 and 9 of the algorithm are converging in nature. The number of processes to consider is limited. For a process network model, their number is usually less than ten.

7. EXPERIMENTAL METHODOLOGY AND SIMULATION OVERVIEW

SrijanSim[2] is a cycle accurate multiprocessor VLIW simulator which is parameterized and is flexible to a large extent, allowing to vary common architectural parameters and plug in new components as needed. Detailed execution statistics for the application are required. SrijanSim is a simulation infrastructure developed for a retargetable framework for cycle-true simulation and analysis of Heterogeneous Multiprocessors. A HMP can be composed by assembling various processing elements, memories, peripheral devices and interconnects. SrijanSim defines modeling methodology to develop simulation models of these components and provides software utilities to facilitate the model development, and includes a rich component library of parameterized and cycle accurate simulation models of memories, VLIW processor, shared bus and non-intrusive thread-aware function-level profiler. Processors are modeled through sequential composition of modules.

7.1 Simulation: Obtaining PN characteristics

The SrijanSim Profiler, a non-intrusive thread-aware function-level statistics collector, can be used to generate a comprehensive profile of multi-threaded applications. The profiler outputs detailed information which can give the designer a keen insight into the application characteristics. This profiler is more elaborate than the standard GNU profiler in a number of ways, the most important advantage being the ability to profile multi-threaded applications to the same level of detail as that of single thread applications. The SrijanSim profiler provides a completion to the extendable and retargetable compiler-simulator infrastructure of Srijan, and establishes an elaborate workbench of integrated tools for research activities focused on application specific multiprocessor architectures.

The profiler yields different profiles such as Macro profile, Flat profile, and Call graph, briefly discussed here. These are generated for each processor in the architecture.

An example of the macro profile of the producer consumer application is shown in table 3 . This is the top level information for all threads in the processor. It gives the total execution time taken by the target application (mapped to that processor). Note that all times have been expressed as a percentage of total application time.

	S_1	S_2	S_3	S_4	S_5	S_6	S
Worst Case	98.4	98.4	98.4	98.4	98.4	98.4	Sum
Corresponding T	586933	7735995	3015712	200344	3211702	2026160	16776846
Algorithm(Step 1 to 5)	96	96	96	96	96	96	Sum
Corresponding T	584812	7720957	3014040	200091	3211421	2025581	1643690
Algorithm(Step 7 to 10)	95.5	94	95	96	96	97	Sum
Corresponding T	584488	7707222	3004040	200091	3211421	2025750	16733012

Table 2: Finding a Better Distribution

Time in %	sspnMPRead rdBf rdCh com	sspnMPWrite wrBf wrCh com	Comp in %	CS in %	Name
0.0	0.0 0.0 0.0	0.0 0.0 0.0	0.0	0.001	ABS
1.7	0.0 0.0 0.0	0.1 0.0 0.1	1.5	0.049	_run_proc_hdr_pn_node1
35.6	0.0 0.0 0.0	1.8 0.0 2.0	31.2	0.176	_run_proc_slice_dec_pn_node1
17.7	1.7 0.0 1.4	1.4 0.0 1.0	10.8	0.112	_run_proc_idct_pn_node1
10.5	0.2 0.0 0.4	1.3 0.0 0.2	8.1	0.089	_run_proc_pred_pn_node1
22.3	2.9 0.0 1.8	2.2 0.0 0.4	14.8	0.087	_run_proc_add_pn_node1
12.1	2.2 0.0 0.5	0.0 0.0 0.0	9.2	0.088	_run_proc_store_pn_node1

Table 3: MacroProfile

As can be seen from the profile besides the six thread corresponding to the six processes, there is also the *ABS* thread, which is the main simulator thread. which spawns the other threads. The following thread level information is listed for each individual thread:

- Name (of the main function) of the thread
- Execution Time for each thread.
- Time spent in reading and writing onto buffers.
- Time spent in computation alone
- Context switching overhead for each thread.

Flat Profile: This profile shows more elaborate information such as the computation time for the application in each function, the number of times that function was called, the context switch overhead for each function, etc. The information listed in this profile allows the designer to infer context switch overheads, program hotspots, inefficient code and other such information.

Call Graph: An example of a part of the call graph of the same application is shown in table4. For each function, the call graph shows which other functions is called, and how many times. There is also an estimate of how much time was spent in the subroutines of each function, as a percentage of the total time in the function. For each thread, a table is displayed, which has an entry for each function. Each entry in this table may have multiple lines. The first line in each entry describes the current function information The subsequent lines of an entry describe the function's immediate children. If the function does not have any children, then no child information is displayed.

7.2 Experimental Methodology

The complete sequence of steps done for obtaining the QoS for individual processes is as follows:

1. Extract video stream containing I frames(single frame streams) and video streams containing known number, 1 I frame and 'n' P frames.
2. Profile to know the execution times of various processes of the application.
3. From the set of data and specified QoS for total application, find the QoS for individual processes
4. Minimize the Total Computation time, respecting the overall QoS.
5. For each process, find the computation time corresponding to the QoS obtained.

8. RESULTS AND DISCUSSION

The application of these steps to MPEG2 Decoder's six processes for S_{req} = 90% is shown in table 2. The columns 2 to 7 of the table correspond to processes P_1 to P_6, while column 8 contains QoS and computation time for total application. The rows contain the S_i and their corresponding T_i. However the T for the application corresponds to $\sum_i T_i$.

T_{min} for the application corresponds to 16652312 cycles for T at 90% QoS. T_{max} corresponds to $\sum T_i$ where T_i corresponds to S_i being 98.4% is 16776846 cycles. By applying the above steps, we obtained a reduction in T, such that it finally becomes 16733012 cycles, which is much closer to T_{min}. The method has resulted in reduction of T, such that it is closer to T_{min} by a factor of $T_{max} - T/T_{max} - T_{min} = 34.5\%$.

Steps 1 to 6 correspond to the basic algorithm. The last steps (7 to 10) are optimization steps and may be omitted if the T_{min} and $T_{initial}$, obtained with equal S_i for the application are not significantly different.

The approach that we have proposed is better than a naive implementation of QoS as a hard constraint, e.g. in our application allowing the processes to run to completion and just maintaining an overall real time constraint of 1/30 frames per second would be the simple solution.

Imposing QoS for each process has two potential advantages. First, as explained in section 2, the use of QoS has been used to lower the HW cost in multiprocessor synthesis. This is achieved since the computation power required for each process is lowered now. The same might be true of resources like network throughput and memory bandwidth required, but we have not measured these. Second, an important area in which optimal QoS distribution

Time(%)	Time(cycles)	Self(%)	Child time(%)	Calls	Name
100.0	726442	0.3			_start
			99.7	1	_sspnOnly
99.7	723999	0.0			_run_proc_hdr_pn_node1
			100.0 1	_proc_hdr	
99.7	723945	0.0			_proc_hdr
			99.7	1	_Decode_Bitstream_proc_hdr
			0.0	1	_next_start_code
			0.2	1	_Initialize_Buffer
			0.0	1	_strlen
0.0	193	100.0			_Initialize_Sequence_proc_hdr
3.8	27889	0.8			_Headers_proc_hdr
			99.2	5	_Get_Hdr_proc_hdr
99.4	721952	0.1			_Decode_Bitstream_proc_hdr
			2.3	2	_Headers_proc_hdr
			97.6	1	_video_sequence_proc_hdr
97.0	704623	0.0			_video_sequence_proc_hdr
			0.0	1	_Initialize_Sequence_proc_hdr
			1.6	3	_Headers_proc_hdr
			0.5	1	_write_seq_prop_proc_hdr
			97.8	3	_Decode_Picture

Table 4: CallGraph

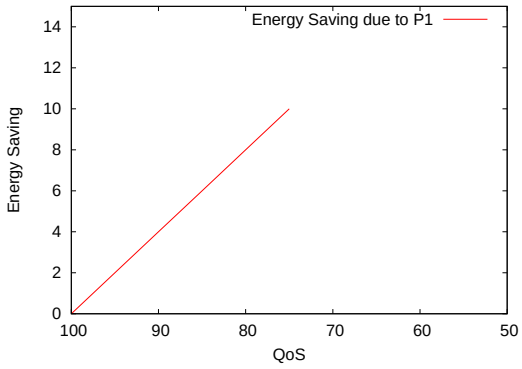


Figure 9: Minimum energy saving by QoS distribution, Energy Saving and QoS expressed in %

can work is energy saving. Dynamic power management works by switching of the processes which follow an already failed process, e.g. in MPEG2 decoding process P_1 is followed by P_2 and so on, and if P_1 fails for a particular frame then P_2 to P_6 need not be executed. In the simple implementation of termination with overall time constraint, processes would be given as much time to execute as they want, with a check only in the end. Say P_1 executes for a long time for one data input, other processes would not be able to meet the overall deadline. But still some of the processes did run to completion. This would result in a higher energy consumption than the optimal QoS distribution, in which P_1 is given just the right amount of time that it needs, and if it fails, we can decide not to execute the other processes, and thus saving energy.

From the results obtained, the number of failures caused by P_1 alone are of the order 4% when QoS total is 90%. Hence, if QoS for individual processes is considered and process is terminated at failure of P_1, then simple calculations on the data show that 96% energy is consumed compared to what would be if processes are allowed to execute having no individual deadlines. Figure 9 shows the energy saving only due to P_1 for different QoS. The X axis shows the total achieved QoS and the Y axis shows the energy saving made, obtained by considering that the cases when processes are switched off, thus saving energy. By considering the other processes, and extending the idea of aborting the iteration of the application as soon as one process fails to meet its individual deadline, the energy saving can be substantially increased.

Energy reduction can actually be made the objective while searching for a solution to $\{S_i\}$. For example in our application consisting of processes P_1 to P_6, QoS of P_1 may be reduced more. Imposing a tighter bound on P_1, implies early identification of failure cases, and hence greater energy saving by following dynamic power management techniques. The algorithm ensures that the QoS requirements are met at every stage, and hence the solution obtained is a more power efficient solution with the same performance. Figure 5 also shows that when lower QoS is accommodated, the power saving made by this approach are more. This is in line with the fact that in low power battery devices, QoS required under low battery conditions would be substantially smaller. In such a scenario, the distribution of QoS with energy reduction as objective will be more suitable.

9. CONCLUSION

Our work shows how QoS specification can be used for efficient architectural synthesis. Video decoding applications have QoS constraints to be met. The requirement was to find the optimal QoS for individual processes constituting the application during the architectural synthesis stage itself. The effect of QoS of individual processes on the overall QoS is application dependent, but we have given a methodology following which, this relation can be found and used for any application. We developed an algorithm which accepts the data of the computation times of individual processes and the required QoS. The output is the set of individual QoS for the processes. The algorithm is based on static analysis of the application and improves the mapping solution, without any dynamic overhead or increase in computation time of the application. The computation times of the processes constituting the application can be obtained using either simulation or estimation.

The solution has been found with the objective of minimizing the

computation time of the application. Currently, the search that has been implemented is exploratory in nature and can be replaced by better techniques. It is also possible to target energy minimization, rather than time minimization, while finding individual constraints for processes.

The experiments were conducted on a PN model of MPEG2 Decoder and the results applied to its mapping. The methodology can be used for application models like periodic task graphs, and the results be used in other mapping and partitioning algorithms. The idea of incorporating QoS can also be extended to other resource requirements like memory and interconnect. Energy aware mapping and partitioning are active areas in research and we have shown that QoS specifications can lead to significant reductions in energy consumption.

10. REFERENCES

[1] Nicola Cranley and Liam Murphy, "Adaptive Quality of Service for Streamed MPEG4 over the Internet," in *ICC2001, IEEE International Conference on Communications*, Helsinki, Finland, June 2001

[2] M.N.V. Satya Kiran and Abhishek Marwah,"SrijanSim: A Retargetable And Efficient System Simulation Framework," in *Technote, Indian Institute of Technology, Delhi*, May 2004

[3] Jeanette P. Schmidt, Alan Siegel and Aravind Srinivasan, "Chernoff-Hoeffding Bounds for Applications with Limited Independence," in *SIAM Journal on Discrete Mathematics*, 1995

[4] David A. Karr, Craig Rodrigues, Joseph P. Loyall, Richard E. Schantz, Yamuna Krishnamurthy, Irfan Pyarali and Douglas C. Schmidt , "Application of the QuO Quality-of-Service Framework to a Distributed Video Application," in *Third International Symposium on Distributed Objects and Applications (DOA'01)*, September 17 - 20, 2001

[5] Ralf Steinmetz and Max Mhlhuser, "Multimedia-Systems: Resources and Quality of Service (QoS)," in *www.cs.odu.edu/ cs778/ralf*, 2005

[6] Radu Marculescu and Amit Nandi, "Probabilistic Application modelling for System Level Performance Analysis," in *Proc. Design, Automation and Test in Europe Conf.*, Munich, Germany, March 2001.

[7] Radu Marculescu, M. Pedram and J. Henkel, "Distributed Multimedia System Design: A Holistic Perspective," in *Proc. Design, Automation and Test in Europe*, France, Feb, 2004.

[8] Jiong Luo and Niraj K. Jha, "Low Power Distributed Embedded Systems: Dynamic Voltage Scaling and Synthesis," in *Proc. International Conference on High Performance Computing (HiPC)*, December 2002.

[9] Basant Kumar Dwivedi, Anshul Kumar and M. Balakrishnan, "Automatic Synthesis of System on Chip Multiprocessor Architectures for Process Networks," in *Intl. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS 2004)*, Stockholm, Sweden, September 2004

[10] Jan Jonnson, " Robust Adaptive Metric for Deadline Assignment in Heterogeneous Distributed Real-Time System," in *Proc. IEEE International Parallel Processing Symposium*, Puerto Rico, USA, April 1999.

[11] G. Kahn, "The Semantics of a Simple Language for Parallel Programming," in *Proc. IFIP Congress 74. North Holland Publishing Co, 1974*.

[12] E. de Kock, G.Essink, W. Smits, P. van der Wolf, J.-Y. Brunel, W. Kruijtzer, P. Lieverse, and K. Vissers, "YAPI:Application Modeling for Signal Processing Systems" in *Proc. Design Automation Conference (DAC-2000)*, pages 402–405, June 2000.

[13] Sorin Manolache, Petru Eles and Zebo Peng, "Optimization of Soft Real-Time Systems with Deadline Miss Ratio Constraints," in *10th IEEE Real-Time and Embedded Technology and Applications Symposium*, Canada, May 2004.

[14] Basant K. Dwivedi, Jan Hoogerbrugge, Paul Stravers and M. Balakrishnan, "Exploring Design Space of Parallel Realizations: MPEG-2 Decoder Case Study," in *International Symposium on Hardware Software Codesign*, Copenhagen, Denmark, April 2001.