A Generic Platform for Estimation of Multi-threaded Program Performance on Heterogeneous Multiprocessors

Aryabartta Sahu, M. Balakrishnan, Preeti Ranjan Panda
Department of Computer Science and Engineering, Indian Institute of Technology Delhi
Hauz Khas, New Delhi, India, 110016, Email: {asahu,mbala,panda}@cse.iitd.ac.in

***Abstract*—This paper deals with a methodology for software estimation to enable design space exploration of heterogeneous multiprocessor systems. Starting from fork-join representation of application specification along with high level description of multiprocessor target architecture and mapping of application components onto architecture resource elements, it estimates the performance of application on target multiprocessor architecture. The methodology proposed includes the effect of basic compiler optimizations, integrates light weight memory simulation and instruction mapping for complex instruction to improve the accuracy of software estimation. To estimate performance degradation due to contention for shared resources like memory and bus, synthetic access traces coupled with interval analysis technique is employed. The methodology has been validated on a real heterogeneous platform. Results show that using estimation it is possible to predict performance with average errors of around 11%.**

I. Introduction and motivation

Embedded applications in the image processing as well as media domain are highly parallel and compute intensive. Further, attempts are being made to include more and more applications in a small size, battery powered embedded device. The implication is that millions of lines of application code have to be ported on to these embedded devices. In such a situation heterogeneous multiprocessors are the only solution to achieve the required performance within the limited power budget. Many factors fuel the use of multiprocessors in embedded systems. These includes complexity and cost of ASIC design, saturation of performance enhancement for single processor platform, significant available parallelism in embedded applications, power consumption issues with higher operating frequencies and VLSI technology offering high integration density.

During the initial design phase, simulation is a widely used methodology to explore the large design space to identify the appropriate components and their configurations, that would meet the required performance "optimally". Simulation takes not only huge amount of time but also requires a complete target compiler tool chain. Generation of target specific tool chain and simulator for heterogeneous multiprocessors is time consuming, especially in the initial design space exploration phase. This is because the design space in terms of configurations of processor, memory and interconnection network is large, and exploring through simulation is not feasible. Alternatively, one can use software estimation tools to estimate performance with considerably less effort (2 to 3 order of magnitude) than simulation, but the accuracy could be an issue [1].

Pthread, OpenMP, PVM and MPI are the widely used parallel programming tools that use either shared memory or distributed memory parallel programing model. In both cases, communication, synchronization and management constructs are written in high-level C language. It is possible to characterize and estimate the execution times of these constructs on the target processor and use this to predict the overall performance.

Estimation of execution time involves analyzing the impact of application mapping over the architecture. Mapping involves defining binding of application components like task, data and communication onto architectural resources like processor, memory and bus/link. Significant complexity arises due to the dynamic nature of this analysis which is completely dependent on mapping and the resulting interaction. The model and estimation technique should be able to capture overheads by identifying performance bottlenecks in the system due to shared resources. This could enable a better mapping/solution. In case mapping changes are inadequate for meeting the performance objectives, transformation of application to enable good mapping as well as changing multiprocessor hardware configuration parameters may be necessary. It is clear that the latter alternatives are time consuming and thus expensive to implement.

The main contribution of this paper is a novel methodology for multiprocessor software estimation and its use in design space exploration. The methodology incorporates "retargetable (i.e. user specified target architecture) estimation" of multi threaded application on user-specified target processor with basic compiler optimizations and a light weight memory simulation. As a part of our methodology, we have used retargetable uni-processor estimator to estimate the communication and synchronization delays by instrumenting the Pthread and other communication library functions. Further it generates target machine dependent synthetic memory and bus trace to estimate delays due to shared resources. It also uses interval analysis as an effective alternative to queuing simulation to estimate resource contention delays.

This paper is organized as follows: in Section II, we give a review of the previous work on multiprocessor performance estimation. In Section III, we present our inputs, outputs and overall methodology of performance estimation of multiprocessors. Section IV describes software behavior estimation of mapped task on a specific processor. In Section V, we describe the method to estimate communication and synchronization delays of multi-threaded tasks. Section VI presents estimation of resource contention related delays and its analysis. Sections VII & VIII contain experimental results, conclusion and future work.

978-3-9810801-5-5/DATE09 © 2009 EDAA

II. Previous work

Performance estimation methods using complete simulation as well as address trace simulation have been studied by Zheng et al. [2] and Kurc et al. [3]. A complete functional simulation approach and trace driven simulation takes considerable amount of time even for a small application. Performance estimation of applications mapped onto multiprocessors using Lost Cycle Analysis (LCA) method is introduced by Crovella et al. [4]. Lost cycle analysis refers to cycles lost in synchronization, communication and resource contention. It also addresses the issue of completeness of estimation using lost cycle analysis. The approach is restricted to a target platform with a set of identical processors connected in a network. The focus is on applications where communication delays are significant.

Mean value analysis (MVA) is a probabilistic method that is used to evaluate design choices for shared bus multiprocessor systems in a throughput-oriented environment. This is described in Chiang et al. [5]. Using MVA technique, performance analysis of subsystems has been done and used for identifying bottlenecks in the architecture (Lee et al.[6]). Performance analysis of heterogeneous multiprocessor multi-cluster system by analytical techniques using probability values has been carried out by Javadi et al. [7]. The model assumes clusters are loosely coupled with some network delays whereas, similar processors inside the cluster are tightly coupled. Probabilistic analytical techniques have limited applicability in embedded domains because the target application can be better characterized and analyzed even at finer granularity.

Snavely et al. [8], presented a performance modeling methodology based on system peak-performance metrics and analysis of application performance by convolving machine signatures (computing power of processor) with application profiles. They used the memory access pattern (MAPS) and machine signature for estimation. Main objective of their analysis is distributed memory message passing model targeted onto supercomputing domain. They do not consider the shared memory model and neither target heterogeneous chip multiprocessors.

We believe our methodology differs from previous works in the following respects and this make it more useful and realistic: we have considered (a) heterogeneous multiprocessor, and the processors are specified in customized high level machine description, (b) taken complete Pthread library and other communication library for calculation of communication and synchronization delays and (c) delays due to shared resource contention are analysed at user specified granularity by abstracting the resoruce access traces using interval analysis technique.

III. Performance estimation methodology

The generic software estimation process described here is driven by three inputs: application specification, architecture specification and partition/mapping description. Application specification is nothing but a fork-join task graph representing a multi-threaded application, whereas, architecture specification refers to a high level description of heterogeneous multiprocessor architecture. Individual processor, network and memory are represented in a machine description language. Mapping information of application components onto architectural components is described separately.

A. Application specification

Application can be represented as a set of tasks forming a fork-join task graph [9] [10]. A fork-join task graph represents parallel phases of computation (forking of tasks) separated by full barrier synchronization (joining of tasks). Pthread and OpenMP standard follow this kind of model. A linear time scheduling algorithm for fork-join task graph exists [10], and it has been shown that such graphs are performance predictable [9]. There are many real life examples from media and processing domain that have been represented in fork-join task graph model. These include Radar beam former, Mpeg2, GSM decoder, 3GPPP radio access protocol (physical layer), Software defined radio, JPEG, LU decomposition, FFT, Merge sort and Bitonic sort [11], [12].

B. Architecture specification

Multiprocessor architecture specification is the second input to the estimator. This contains descriptions of processors, network and memory. The multiprocessor architecture and its components are represented in high level machine description. The architecture description consists of a three-tuple (P, M, L) where P is the set of heterogeneous processors. The heterogeneity of the processors can be in terms of frequency, functional units, pipeline stages etc. M is the set of memory units and L is the set of links between processors and memories. These can be a shared bus or individual dedicated bus. In our methodology we have used HMDES description [13], [14] to describe P, M, and L, which makes the platform description flexible and generic.

C. Partition description

Application partition in heterogeneous multiprocessors can be defined as mapping of application three-tuple (task, data, communication) onto architecture three-tuple (processor, memory, link or channel). Symbolically it can be represented as (T, D, C) mapped to (P, M, L). Application performance over an architecture not only depends on the architectural components but also on the partition and/or mapping. The main objective in design space exploration is to explore different mappings of application components on to architectural components.

D. Performance estimation methodology

Performance estimation for a given application partition over a specific architecture is composed of the following phases

- Retargetable estimation of individual task execution time on the mapped processor, independent of other tasks and communications: Uni-processor software estimation technique was used to do this work (details in Section IV).
- Retargetable estimation of communication and synchronization delays as well as dependency delays: A C-language library for communication and synchronization tasks is available, and as these library components execute on some processor, so the overhead due to these also be estimated using the software estimation technique (details in Section V).
- Shared resource contention analysis and estimation of associated delays: Requests from various sources are jointly analyzed to estimate the delays due to shared resources (detail in Section VI).
- Estimation of execution time: All the delays and overheads are combined to generate the expected execution time.

mult by shft	*array-ref mul red*	*remo array indx by ptr*	*remove reg conv*	*ptr incr pass*
mul i,4	lda r1 A mul r2 i n add r2 r1 lod r3 O(r2) lda r1 A	lda r1 A mul r2 i n add r2 r1 lod r3 O(r2)	lda r1.s i cvt r2.s r1.s add r3.s r5.s cvt r4.s r3.s mul r5.s r3.s	mov r1 A ldc r2 4 add r2 r1 mov A r2
⇓	⇓	⇓	⇓	⇓
sl i, 2	lda r1 A sl r2 i lg(n) add r2 r1 lod r3 O(r2)	add A A 4 lod r3 O(A)	lda r1.s i add r1.s r5.s mul r5.s r1.s	add A 4

TABLE I
MEANING OF COMPILER OPTIMIZATION

IV. ESTIMATION OF EXECUTION TIME OF TASK ON MAPPED PROCESSOR

Performance estimation essentially refers to predicting the execution time of a mapped individual task on a specific processor. To estimate multiprocessor behavior with shared resources like memory and bus, we also need the resource access pattern of the task when it is executed on the processor. Inputs to the estimator includes processor description and C description of the task. "Processor signature" can be defined as computing power of processor independent of application [8]. Processor is described in a high level machine description (HMDES) which captures the processor signature. This description includes the number of pipeline stages, issue-width, register file (RF) size and available functional units (FUs) and their functionality. "Task profile" can be defined as detailed listing of the primitives and operations to be carried out by the task independent of the target processor on which it is executed. Operations of interest in our estimation include primitive operations like integer operations, floating point operations, branch operations and load/store operations.

A. Estimation flow

Performance can be estimated by simply list scheduling primitive operations on the processor resources. Total execution time can be estimated as

$$T_{exe} = \sum_{bb=1}^{N_{bb}} L[bb] \times F[bb] \quad (1)$$

Where $L[bb]$ is latency of bb_{th} basic block, $F[bb]$ is the expected/profiled frequency of bb_{th} basic block and N_{bb} is the total number of basic blocks in the task. Latency of a basic block can be found by using list scheduling basic block instructions on the FUs of the mapped processor. SUIF (Stanford University Intermediate Format)[15] compiler framework is used for this purpose. Figure 1 shows the overall framework for the estimation of execution time on a uni-processor.

SUIF compiler generates flat unoptimized assembly codes. In performance estimation it is necessary to include some basic compiler optimizations. These basic compiler options include conversion of index array access to pointer array access, data type conversions, replacement of multiplication by a constant to a set of left shift operations, pointer increment, loop index increment and global register allocation. These compiler options are specified as part of processor description and act as directives for the estimation program. In this way, it mimics the role of cross compilers. Examples of some of the compiler optimizations, where effect have been estimated in our approach are given in Table I. Part of the HMDES file combining these directives is given in Figure 2. Profiling of basic blocks at

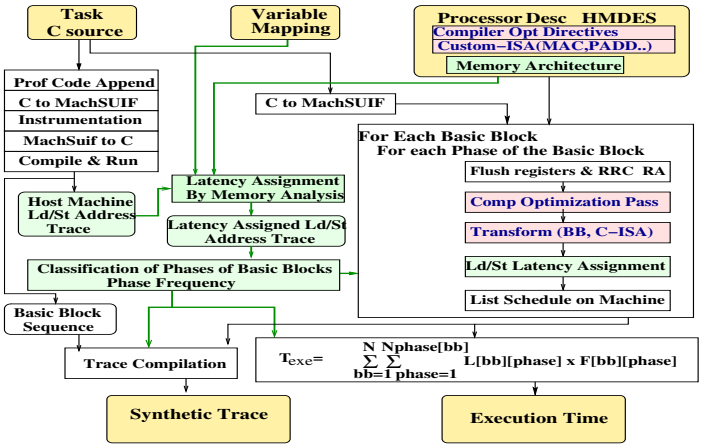


Fig. 1. **Framework uniprocessor performance estimation.**

```
Section Compiler_Optimization_Options
{
const_mult_by_shift_and_add      On   remove_register_conversion  Off
array_reference_mult_reduction   On   increment_pass              On
remove_index_by_ptr              Off  local_register_allocation   Off
ptr_increment_pass               On
}
```

Fig. 2. **Optimization options as part of HMDES.**

machine level is shown on the left side of Figure 1. The input source code was instrumented using SUIF. The instrumented C code was run on a host machine to get profile data. The profile data includes basic block frequencies, basic block sequence and load/store address trace. Latency assignment to each load-store request was done by a light weight memory simulation. Memory simulation takes three inputs; address trace, variable mapping to memory and memory architecture. Based on latencies of loads/stores of a basic block, each instance of basic block are classified into one of the many phases of execution. These phases of execution of basic block can differ in load-store latencies for the same address depending of the nature of access e.g. cold cache, write, scratch pad access etc. Execution time estimation using profile data and processor resource is shown on right side of Figure 1. For each basic block, The estimation process includes register allocation using register reuse chain algorithm[16], transformation of basic block for machine compatibility, latency assignment and list scheduling on processor resources.

B. Generation of shared resource access pattern

Light weight memory simulation tries to annotate a latency number to each memory reference. The list scheduler uses this memory latency to annotate cycle information for each operation in the basic block. Sequence of execution of basic blocks, list scheduled basic blocks together with cycle time information are sufficient to generate the cycle annotated memory trace. This task can be accomplished at the time of estimation of execution time. In a similar manner traces for other shared resources has been generated.

C. Uniprocessor estimation result

The components of total execution time are computation time (integer, float), memory delay, control delay and dependency overhead. For accurate estimation, it is necessary to take care of all such latencies including their potential overestimation (due to overlap) and underestimation (due to stalling or resource

P→	Cradle PE			Leon3 with FPU			SS-mips		
B↓	Act	Est	%E	Act	Est	%E	Act	Est	%E
dct	98563	104167	-5.6	43947	42539	3.2	53852	50506	6.2
lu	23624	26990	-14.3	10004	10192	-0.1	11864	12061	-6.1
fft	16128	13823	14.3	8184	6335	22.6	6321	5782	8.5
mat	92258	94250	-2.1	40436	38779	4.1	54643	45115	17.4

TABLE II
ESTIMATED AND ACTUAL EXECUTION CYCLES

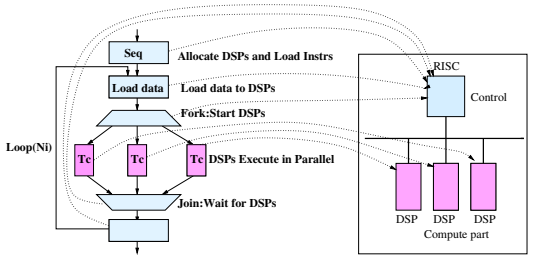


Fig. 3. **Parallel DSPs execution initiated by a RISC.**

contention). Our model is able to capture all the delay components individually and also account for the perturbation due to their interference. Table II shows the estimation results for three different processors namely Leon3 (7 stage pipeline, multiport RF, with FPU) [17], CradlePE (Non-pipeline) [18] and ss-mips (simplescalar 5 stage pipeline) [19]. Results show an average error of around 14% between actual and estimated performance.

V. Performance of Multi-Threaded Tasks

Applications are represented as fork-joins of tasks and implemented in Pthread kind of parallel programming tool. Communication, synchronization and management constructs are written in high-level C language and it executes on one of the processors. So it is possible to characterize and estimate the execution times of these constructs on the target processor and use this to predict the overall performance. In this work, we have estimated the behavior and execution time of Pthread library and other communication library using SUIF retargetable uni-processor estimation framework as described in Section IV.

In many multiprocessor architectures, one or more RISC processors are used for I/O control, scheduling, as well as synchronization along with a number of DSPs for mapping the compute intensive tasks. Cradle CT3400 [18] is one such architecture, which has been modeled in this work for performance estimation. Considering such an architecture, the active processor is waiting for computation to finish in all the DSPs. Time to run parallel threads in DSPs as shown in Figure 3. can be given by the following equation.

$$T_{fg_d} = N_d\ (t_a + t_{li}) + N_i\ (N_d\ (t_{lr} + t_s) + t_{c_d} + \gamma_r\ ) \quad (2)$$

Where N_d is the number of DSPs used, N_i is the number of iterations, t_{lr} is time to load the data to DSP, t_{li} is time to load the instruction to instruction memory of DSP, t_s is start up time of DSP from PE, t_a is time to allocate a free DSP and t_{c_d} is the computation time of function in DSP. γ_r is delay due to shared resource contention (detail about this delays is described in Section VI). C sources of library of communication constructs, library of synchronization constructs and parallel task are mostly available. All these constructs and tasks execute on some processor. So the values of t_{c_d} t_{lr}, t_{li} and t_a can be estimated using retargetable uni-processor software estimation framework as described in Section IV. Time for the whole

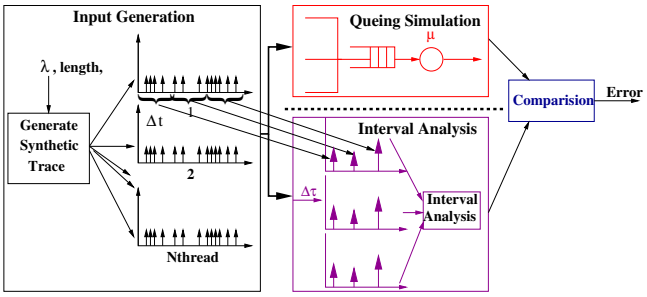


Fig. 4. **Experimental framework for interval analysis.**

application can be estimated by summing up the time for all fork-joins along with some sequential pre and post processing. Normally a complete application is composed of nested fork-joins. And in such each fork-join, a forked path is assigned to a processor. A fork-join may also be iterated for a number of times. The total execution time can be estimated as

$$T = T_{pre} + T_{post} + \sum^{NestedSum} (T_{sq} + \sum_{i=0}^{N_{fg}[i]} T_{fg}[i]) \quad (3)$$

Where T_{pre} and T_{post} are pre-processing and post-processing time respectively for using a single PE and $N_{fg}[i]$ and $T_{fg}[i]$ is the iteration count and time taken respectively by the i_{th} fork-join.

VI. Resource Contention Analysis of Fork-Join

Resource contention delays depend upon the number of processors executing in parallel, request rate of the task executing on them, contention resolution policies and bandwidth of the shared resource. Typically shared resources in the system are bus, lock, cache, scratch pad memory and DRAM. If access request patterns of resources are available, than all additional delays due to contention for a shared resource can be estimated by first timed summing up of requests for the shared resource followed by interval based bandwidth analysis. Shared resource access pattern from a single task is the trace of access to a shared resource when a task gets executed on a mapped processor. In this section, it is shown that "interval analysis is an effective alternative solution to queuing simulation" to estimate the resource contention delay.

A. Interval analysis: An abstract interpretation

The approach presented here treats individual request streams as a "queue" and uses interval analysis to generate effective access rate data. To some extent, accuracy can be traded off with analysis time by changing the size of interval. Thus, interval queuing analysis in this context implies analyzing the trace. One basic parameter required for analysis is the size (in time) of the interval. It is easy to realize that the analysis accuracy improves by reducing the interval period while analysis time goes up. To understand this trade off, extensive experimentation with synthetic traces were carried out. Our experimental framework for interval period - analysis time trade off is shown in Figure 4. In this work, we have generated synthetic traces for N threads with each thread having its own request rate distribution. Assuming there is a single server, all threads are fed to a queuing simulation as well as to interval analysis module.

B. Delay calculation in interval analysis

For simplicity of analysis and understanding, lets us assume there is a single request source per processor and there is a

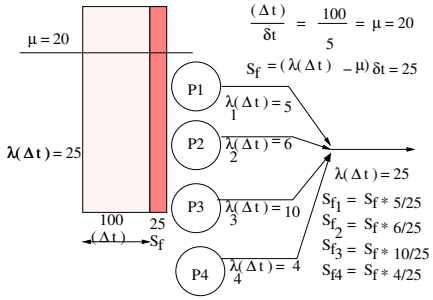


Fig. 5. **Spreading of interval.**

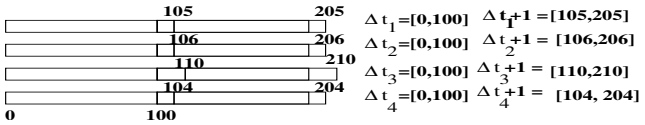


Fig. 6. **Time references for next interval.**

single shared resource. The processors are generating load/store request with an arrival rate λ and the requests are being served at the rate μ. Typically, processors cannot generate further requests if a pending request is not served completely (blocking request). This implies that a processor can have at most $\frac{\Delta t}{\delta t}$ number of requests in the time interval Δt. So $\lambda_i(\Delta t) \leq \frac{\Delta t}{\delta t}$. Total requests to a shared resource is sum of requests from all processors

$$\lambda(\Delta t) = \sum_{i=0}^{p} \lambda_i(\Delta t) \tag{4}$$

There will be a contention in a particular time interval if $\mu(= \frac{\Delta t}{\delta t}) < \sum_{i=0}^{p} \lambda_i(\Delta t)$. But in a time interval $\lambda(\Delta t)$, all the requests have to absorbed due to blocking behavior of the processor. Thus, these periods will be extended due to congestion as shown in Fig. 5. This corresponds to spreading of the interval. The spreading factor of time interval depends on the present request rate in that interval.

$$S_f(\Delta t) = (\lambda(\Delta t) - \mu).\frac{1}{\mu} \tag{5}$$

The interval spread seen by different processors depends upon the request rates from all the processor and reflects the interaction effect as shown in Figure 5. This is given by

$$S_{f_i}(\Delta t) = (\lambda(\Delta t) - \mu).\frac{1}{\mu}.\frac{\lambda_i(\Delta t)}{\lambda(\Delta t)} = S_f(\Delta t).\frac{\lambda_i(\Delta t)}{\lambda(\Delta t)} \tag{6}$$

For the next time slot, the actual time interval will be different for different processors due to variable spreading as shown in Figure 6. This has to be accommodated for computing the joint request rate. The total delay due to the shared resource contention can be estimated by summing up the maximum of spreads observed by all the processors in each time interval.

$$\gamma_r = Delay_{res\_cont} = max(\sum_{j=1}^{N_{interval}} S_{f_i}(\Delta_j t)) \tag{7}$$

The above analysis assumed equal priority for all requests. On the other hand, priority of request type r from i_{th}processor can also be characterized and the delays can be calculated as follows

$$S_{f_i}(\Delta t)_r = \begin{cases} 0 & if \sum_{k=0}^{Pr} \lambda_k < \mu \\ \mu - \sum_{k=0}^{Pr} \lambda_k & otherwise \end{cases} \tag{8}$$

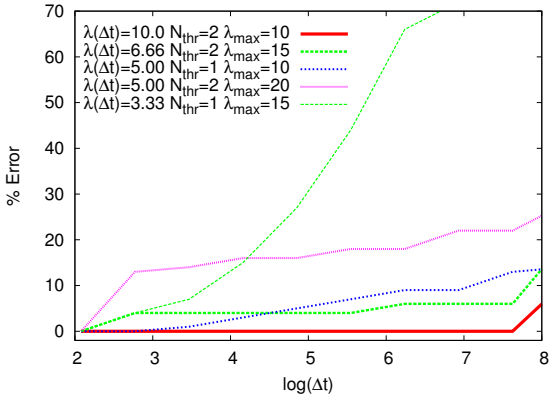


Fig. 7. **Errors of estimation.**

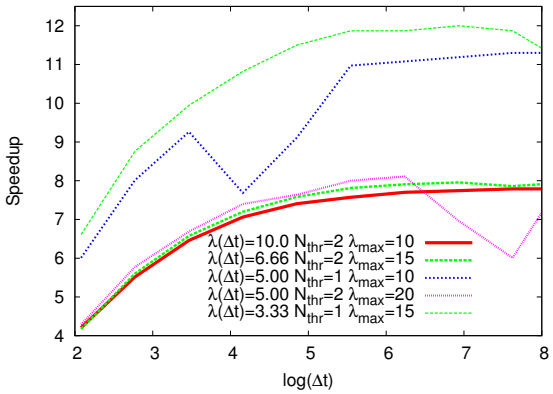


Fig. 8. **Speedup of estimation.**

C. Result of interval analysis

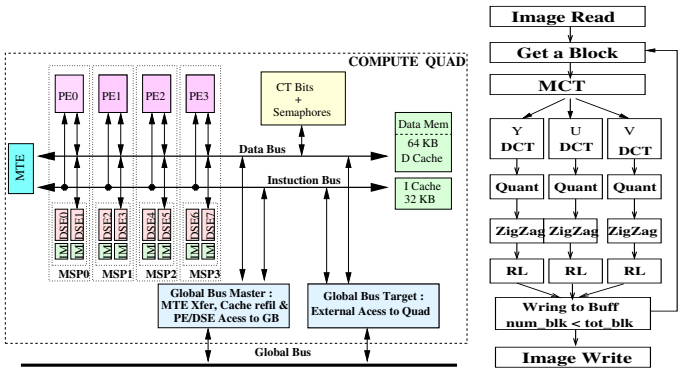
Error and speed up between the queuing simulation and interval analysis are shown in Figures 7 and 8 respectively. In both the figures X-axis corresponds to the size of interval in log scale, where as Y-axis corresponds to % error in Figure 7 and speed up in Figure 8 respectively. The following observations follow:

- Estimation error is marginal in the curve corresponding to high request rate regions ($\lambda(\Delta t) \geq 5$) where most of the contention delays occurs.
- Estimation error is high in the curve corresponding to low request rate regions but delay contribution to the system is much less and can be ignored.

Due to the above factors, we conclude that interval analysis works reasonably well in all situations. From result graphs it is clear that for higher request rates, speeds up to 12x is possible while error in estimation is lower then 7%.

VII. Experiments and Results

For detailed experimentation, we took the Cradle Rapid Development System (RDS)[18] as an example of a heterogeneous multiprocessor platform. This board has a CT3400 heterogeneous multiprocessor chip. As shown in Figure 9a, it consists of 4 RISC processors (PE, 32 bit), 8 DSPs (Digital Signal Engine), 1 four threaded DMA, 32KB Instruction cache and 64KB data cache/scratch pad memory, all are connected on a shared bus. The device is supported by an external board with 256 MB DRAM and gcc compiler for the RISC processors. We have implemented eight different partitions of JPEG application (Figure 9b) and obtained the actual performance numbers. The



(a) Architecture of Cradle CT3400. (b) JPEG application

Fig. 9. **Architecture and application**

t_a	$t_{s_{dse}}$	$t_{s_{pe}}$
4652	5467	1.4e6
$t_{li_{mct}}$	$t_{li_{dct}}$	$t_{li_{zz}}$
15191	16005	15242
$t_{ld_{mct}}$	$t_{ld_{dct}}$	$t_{ld_{zz}}$
8466	12882	6432

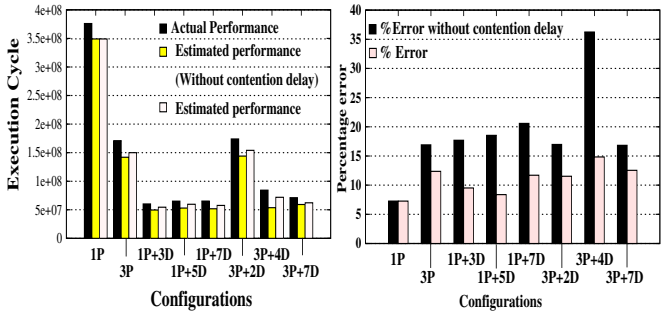
	get blk	mct	dct 1-3	qnt 1-3	ZZ 1-3	RL 1–3
1P	P0	P0	P0	P0	P0	P0
3P	P0	P0	P012	P0	P0	P0
1P3D	P0	P0	D012	P0	P0	P0
1P5D	P0	D0	D123	P0	D4	P0
1P7D	P0	D0	D123	P0	D456	P0
3P2D	P0	D0	P012	P0	D123	P0
3P4D	P0	D3	D012	P0	P02	P0
3P7D	P0	D0	D1-6	P0	P012	P0

TABLE III
(A) COMMUNICATION TIME (B) MAPPING DESCRIPTION

same performance was estimated using our framework. The main motivation behind taking one application is to compare results of several different mapping of application onto the target architecture.

Number of cycles for communication time between PEs and DSE's, allocation of DSE (t_a), starting a DSE ($t_{s_{dse}}$), starting other PE ($t_{s_{pe}}$), loading data (t_{ld}) and instruction (t_{li}) to DSE are shown in Table III(a). The values shown are estimated using retargetable uni-processor estimator, using source code of Cradle thread library and communication library as input. Table III(b) shows the mapping of JPEG application over Cradle architecture. Each column corresponds to a task and each row corresponds to a mapped configuration. 3P4D refers to 3 RISC and 4 DSE configuration. Each entry specifies the binding of the task to one or more processors. D012 in 6^{th} row, 4^{th} column implies, that DCT 1-3 are bound to DSE 0, 1, and 2.

Figure 10a and Figure 10b shows the actual execution cycles and estimated execution cycles of eight mappings of JPEG application over Cradle architecture. Without taking into account the resource contention delays, the errors range from 7.27% to 36.2% with an average error of 17.6%. After adding the



(a) Actual and estimated execution cycles (b) %Error without and with resource contention

Fig. 10. **Estimation result**

delay due to resource contention part of the system (taking $\Delta t = 8 * t_{MemRead}$), the errors reduce significantly and range from 7%to 14%, with the average being around 11.3%.

VIII. Conclusion and Future Work

Performance of the applications mapped onto a multiprocessor architecture depends considerably on mapping or binding of application components onto the architecture resources. In multiprocessor environment, mapping not only influences effective use of concurrent resource but also has a significant impact on delays due to synchronization and resource contention. In this paper we have presented a framework for retargetable performance estimation of multi-threaded applications onto heterogeneous multiprocessors. Experimental results show that the estimation errors are around 11% implying that the methodology can be useful in the first phase of design space exploration. Further, identifying performance bottlenecks to provide feedback to the designer for architecture enhancement is another possible application of this methodology.

Writing and debugging of multi-threaded code is a difficult exercise. Cilk [12], StreamIt [11] and X10 [20] are parallel programming languages, where one can specify the application and its parallelism at algorithmic level or capture the graphical structure of the application. Integration of this framework with Cilk or StreamIt-like high level compiler is being pursued in the next phase.

References

[1] S. Kwon, C. Lee, S. Kim, Y. Yi, and S. Ha, "Fast design space exploration framework with an efficient performance estimation technique," *Embedded Systems for Real-Time Multimedia*, pp. 27–32, Sept. 2004.

[2] G. Zheng, T. Wilmarth, P. Jagadishprasad, and L. V. Kalé, "Simulation-Based Performance Prediction for Large Parallel Machines," *Int. J. Parallel Program.*, vol. 33, no. 2, pp. 183–207, 2005.

[3] T. Kurc, M. Uysal, and et. al., "Efficient Performance Prediction for Large-Scale, Data-Intensive Applications," *Int. Journal of High Performance Computing Applications*, vol. 14, no. 3, pp. 216–227, 2000.

[4] M. E. Crovella and T. J. LeBlanc, "Parallel performance prediction using lost cycles analysis," in *Proc. of ACM/IEEE conf. on Supercomputing*, 1994, pp. 600–609.

[5] M.-C. Chiang and G. S. Sohi, "Evaluating Design Choices for Shared Bus Multiprocessors in a Throughput-Oriented Environment," *IEEE Trans. Comput.*, vol. 41, no. 3, pp. 297–317, 1992.

[6] C.-S. Lee and T.-M. Parng, "A Subsystem-Oriented Performance Analysis Methodology for Shared-Bus Multiprocessors," *IEEE Trans. on Parallel and Distributed Systems*, vol. 07, no. 7, pp. 755–767, 1996.

[7] B. Javadi and J. H. Abawajy, "Performance Analysis of Heterogeneous Multi-Cluster Systems," in *Proc. of the Int. Conf. on Parallel Processing Workshops*, 2005, pp. 493–500.

[8] A. Snavely, N. Wolter, and L. Carrington, "Modeling Application Performance by Convolving Machine Signatures with Application Profiles," in *IEEE Workshop on Workload Characterization*, 2001, pp. 149–156.

[9] V. S. Adve and M. K. Vernon, "Parallel Program Performance Prediction using Deterministic Task Graph Analysis," *ACM Trans. on Computer Systems (TOCS)*, vol. 22, no. 1, pp. 94–136, 2004.

[10] Q. Li, Y. Ruan, ShidaYang, and T. Jiang, "An optimal scheduling algorithm for fork-join task graphs," *Parallel and Distributed Computing, Applications and Technologies*, pp. 587–589, Aug. 2003.

[11] M. I. Gordon, W. Thies, and S. Amarasinghe, "Exploiting Coarse-Grained Task, Data, and Pipeline Parallelism in Stream Programs," in *Int. Conf. on ASPLOS*, 2006, pp. 151–162.

[12] M. Frigo, C. E. Leiserson, and K. H. Randall, "The Implementation of the Cilk-5 Multithreaded Language," in *Proc. of the ACM SIGPLAN conf. on PLDI*, 1998, pp. 212–223.

[13] L. N. Chakrapani, J. Gyllenhaal, and et. al., "Trimaran: An Infrastructure for Research," in *in Instruction-Level Parallelism. LNCS*, 2004, p. 2005.

[14] A. Halambi and P. Grun, "EXPRESSION: A Language for Architecture Exploration through Compiler/Simulator Retargetability," in *In Proc. of the DATE*, May, 1999.

[15] R. P. Wilson and all, "SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers," *ACM SIGPLAN Notices*, vol. 29, pp. 31–37, 1994.

[16] L. Wehmeyer, M. Jain, and all., "Analysis of the Influence of Register File Size on Energy Consumption, Code Size, and Execution Time," *IEEE Trans. on CAD of IC and Systems.*, vol. 20, pp. 1329–1337, Nov 2001.

[17] "LEON3 SPARC V8 Processor Core," http://www.gaisler.com/.

[18] "Cradle Technologies CT3400 Data Sheet," http://www.cradle.com.

[19] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," *Computer*, vol. 35, no. 2, Feb 2002.

[20] S. Agarwal and et al., "May-Happen-in-Parallel Analysis of X10 Programs," in *Proc. of ACM SIGPLAN PPoPP*, 2007, pp. 183–193.