

On Tools for Modeling High-Performance Embedded Systems

Anilkumar Nambiar and Vipin Chaudhary

Department of Computer Science, Wayne State University
{anambiar, vipin}@wayne.edu

Abstract. Most of the new embedded systems require high performance processors at low power. To cater to these needs, most semiconductor companies are designing multi-core processors, also known as chip-multiprocessors, while some are developing multi-chip boards with existing multi-core processors. Developing applications on these powerful architectures require specialized tools to obtain the optimum performance. Most applications running on these processors not only require high processing power, but also have tight resource constraints. For both, chip-multiprocessors and multi-chip boards, one faces some common problems while developing applications for them. To meet this end, we have developed tools to model high-performance embedded applications on these complex high-end systems. We have integrated these tools in the modeling framework of MILAN, and have modeled a real application, Mpeg-2 Audio Video Decoder. For validation we have used Cradle Technologies MDSP multi-core chip as the target processor.

1 Introduction

Chip Multiprocessors are being widely used to develop System-on-Chip (SOC) solutions for high-end embedded systems. An SOC consists of processors (RISC/DSP), internal memory, external memory, device controllers, timers, etc. SOC architecture for a low/medium scale device would not require more than a couple of processors. On the other hand, high-end embedded systems, such as HDTV, bio-medical applications, high capacity printers and copiers would require multiple cores of RISC/DSP or a combination of both, to fulfill the high performance requirement. Most of these applications can be parallelized over multiple low power processors to obtain the desired performance.

Most of the multi-core chips available consist of not more than 8 cores on a single chip [1]. For a high-end application this might not be suitable. ASIC-based solutions are not suitable either, as these solutions take longer time to market and have high cost per unit. There are few companies developing multi-core chips with 30-60 cores in one chip [2] [3]. While other companies, in order to achieve the high performance requirements for these high-end applications, develop multi-chip boards, with each board

housing 2-4 chips, where each chip has 4-8 processors [4][5][6]. This gives enough computational power for the high-end systems.

As the number of processors increase, the bandwidth requirement for the application over the shared global bus increases. In order to sustain such high requirements, these systems have a hierarchical architecture consisting of clusters. Each cluster is made up of RISC and/or DSP processors with some shared internal memory to be used as a scratch pad. In addition, it has semaphores, timers and interrupts. Any access to the resource within the cluster takes less time as compared to any access to resources in the other cluster or global resources since the request has to be served through shared global bus. The desirable implementation on this system would be to reduce access outside the cluster in order to improve performance. The programmer can also benefit from the nature of most of the high-end applications, where the computation to communication ratio is high.

Due to similarity in architecture, these high processor-count chip-multiprocessors and multi-chip board solutions face some common problems for developing applications. Though these systems provide high computational power, they require specialized tools to harness their raw power. One requires tools to design and map resource constraint applications on these high performance systems, tools to simulate the system at various granularity levels and provide feedback through profiling statistics, which can be incorporated back into the design to optimize the application. As most of the high-end system would be heterogeneous, the tools must provide mechanisms or interfaces for seamless integration of different simulators for different types of processors in the underlying system.

There are lots of tools for modeling SOC systems. Some of them are Ptolemy[7], Polis [8], Chinook [9], SystemC [10] and MILAN [11]. We tried to use these tools for modeling systems on such clustered/hierarchical architecture and found that they were not suitable in their existing setup. Modification needs to be done to incorporate the type of architecture we were using. Most of these tools are complex and proved cumbersome to integrate the simulators for this architecture. MILAN is relatively easy to understand and provides simple interface to integrate any new simulator. It also provides Design Space Exploration capability to obtain a near optimum design solution for the system under consideration.

For validation, we have considered Cradle Technologies' MDSP architecture [12]. This architecture uses a combination of low performance RISC and DSP processors, with each chip consisting of 36 to 60 processors. This architecture provides a very high raw computational power in the range of 10-15 GMACs. We have developed an interpreter for MILAN to model applications for MDSP. In addition, we have developed and incorporated a new heuristic for mapping applications on clustered architecture. In this paper, we have presented modeling of a complex application Mpeg-2 Audio Video Decoder (MAVD) using MILAN on the MDSP chip.

The paper is organized as follows. Section 2 gives the problem definition, section 3 discusses the Model Interpreter, section 4 discusses the MAVD modeling, section 5 discusses the related work in this area and section 6 has the concluding remarks.

2 Problem Definition

The MDSP is a heterogeneous multi-core chip developed by Cradle Technologies. The chip has a hierarchical architecture, with clusters within a chip. Each cluster or Quad consists of 4 RISC processors or PEs, 8 DSP co-processors or DSEs, 64 Kbytes of data memory, 4 channel DMA engines and other resources such as semaphores, and timers. Each chip may have varying number of clusters from 1 to 5. If one wants to design an MPEG-2 Audio Video Decoder (MAVD) system using the MDSP chip, one needs to decide on the resource requirements for each task. A typical MAVD system will consist of following tasks: Audio Decoder, Video Decoder, Audio Renderer, Video Renderer, TS Parser, System Controller, TS Feeder Simulator. The memory and processor requirement provided by Cradle Technologies is given in the Fig. 1.

Fig. 1 MAVD System Resource Requirement Table

| Program and its Resources | Audio Decoder | Audio Renderer | Video Decoder | Video Renderer | TS Parser | System Controller | TS Feeder Simulator | ISR Simulator | Timer Simulator |
|---------------------------|-------------------------------|----------------|----------------|----------------|---------------|-------------------|---------------------|---------------|-----------------|
| Computation | 1 PE 1 DSE (1 data DSE) | 1 PE 1 DSE | 6 PE 12 DSE | 1 PE 2 DSE | 1 PE 1 DSE | 1 PE | 1 PE | 1 DSE | 1 DSE |
| Private Local Memory | 0x22D0 | 0xDD0 | 0xD68 | 0xBF8 | 0xED8 | 0x2F0 | 0x78 | - | - |
| Shared Local Memory | - | - | 0x9C0 | - | 0x8 | - | - | - | - |

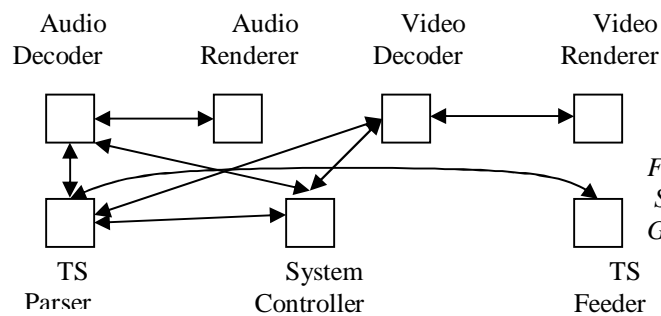


Fig. 2 MAVD System Communication Graph

While designing such a complex system, the extent of communication between different modules might not be obvious. One would like to select a design, which would minimize the execution time. Most of the time the system tasks are developed independently and needs to be integrated. During integration, one would like to map the system on the chip, which would reduce the communication time, which in turn would minimize the execution time. MILAN provides excellent design space exploration functionality, whereby one can consider different designs, simulate those at high level or low level using the appropriate architecture simulators and then choose a good solution. In our case, since we are only considering one chip, the simulation becomes simple. MILAN incorporates an optimal mapping algorithm for linear array of tasks [13]. But it requires the tasks to be executed in linear order, whereas in our case, the tasks are executed concurrently on different processors. Hence it cannot be used for our problem domain. In order to support MDSP chip in MILAN, one needs to provide:

- a. MILAN interpreter to interpret system models and generate output for MDSP simulator, and associated tools.
- b. A new mapping heuristic to map applications on MDSP processor.

3 Model Interpreter

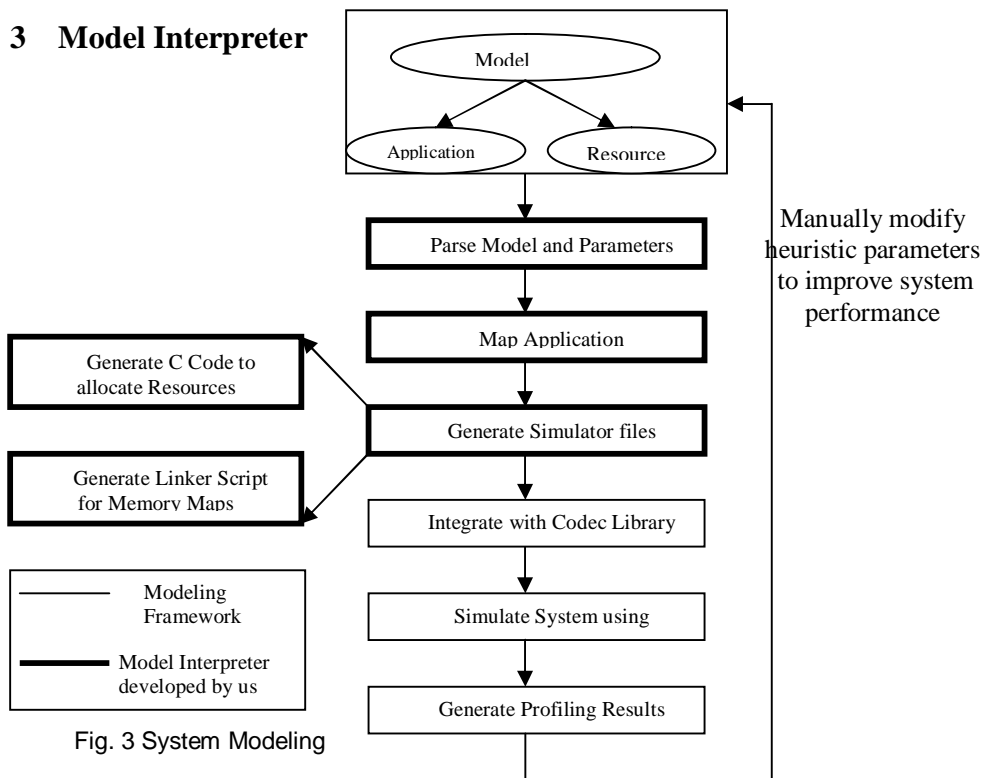


Fig. 3 System Modeling

MILAN uses GME or Generic Modeling Environment [14] to synthesize a domain specific modeling interface for modeling the system. GME provides interfaces to specify different modeling paradigm for different domains. Modeling paradigm is a set of building blocks and composition rules using which, the designer models the system. MILAN is an example of such a modeling paradigm.

While modeling any system, the designer will perform two main activities:

1. Application Modeling, where the designer defines application modules or tasks and inter-task communication between different modules using task-graph.
2. Resource modeling, where the designer models the underlying architecture. In this the designer defines various architecture parameters such as frequency, power consumption, cache size etc. for the architecture under consideration.

In application modeling, we found no provision to specify shared memory communication between multiple modules. In the existing setup in order to represent shared memory communication one would have to show it by connecting different modules using communication channels. This might complicate the design with lots of connections between modules involved in communication. In order to represent shared memory communication, we added a new feature to the paradigm to represent shared memory communication.

MILAN provides various model interpreters to interpret the model and generate output for different simulators. In order to generate the output for MDSP simulator, we developed our own interpreter. This interpreter takes the model and applies our proposed heuristic to generate a good mapping scheme and then generate code to allocate resources and launch the specific threads of various tasks on the processors. This divides the interpreter into three main components, the Model Parser, the Mapping Heuristic and the Code Generator.

3.1 Model Parser

The Model parser is a COM object, which is invoked from the GME as an interpreter. This parser performs a top-down parsing of the components defined in the GME and forms the input to the Mapping heuristics. It forms the list of tasks T defined, based on primitive components. It also forms the resource requirement matrix and communication matrix. The priority for each task is obtained from the attributes of the task. The processor requirement for each task is obtained from the resource reference provided by the user in the Mapping Aspect window. These inputs are provided to the Mapping heuristic module to provide a good mapping scheme.

3.2 Mapping Heuristic

MILAN provides a mapping algorithm to map linear array of tasks onto single and multiple device. It requires the tasks to be in linear order where, given a set of tasks $T =$

$\{t_1, t_2, \dots, t_n\}$, t_{i+1} executes only after t_i , $1 \leq i \leq n$. The mapping problem in our case is different than the one discussed.

Suppose we want to map the MAVD system on the MDSP processor by integrating different tasks, which were developed independently. In this case some of the tasks are independent, while others have inter-task communication. The first requirement would be to allocate all the processors for a given task in the same cluster, as most of the tasks will have more intra-task communication than inter-task communication. In addition to intra-task communication, the tasks will communicate with the allocated resources to varying degree. The resources in this case are memory, and co-processor. If there were more communication between co-processor and processor than inter-task communication, it would be better to allocate the co-processor from the same cluster in which the task is running, and then try to see if the two tasks involved in communication can be allocated to the same cluster.

We have developed Improved First Fit Decreasing [15], which improves over the existing First Fit Decreasing (FFD) algorithm [16]. Since we use a priority queue, if there are m clusters then this will require $O(n \log n * m)$ time, where n is the number of tasks to be mapped. So, in worst case it would be $O(n^2 \log n)$, where the number of tasks is equal to the number of clusters in the chip.

3.3 Code Generation Module

The code generation module generates C code to allocate processors and launch programs on the processors. To compile a multi-task program on MDSP, one needs a loader script also known as *LD* script to determine the code sections and the memory requirement for the executable. This *LD* script is used by the GNU *ld* program to generate the executable. The script specifies the tasks running with information about the local and DRAM data memory used by each task. In addition it also specifies the code sections for each task.

For any program running on MDSP, the main thread is started on QUAD 0, PE 0. This thread will start the other threads on the other processors. The task, which is mapped to QUAD 0, PE 0 forms the main thread. This thread will launch one thread from each of the other tasks on the mapped processors. Those threads in turn will start threads from its task. The launching is similar to a tree structure and its shown in the following figure.

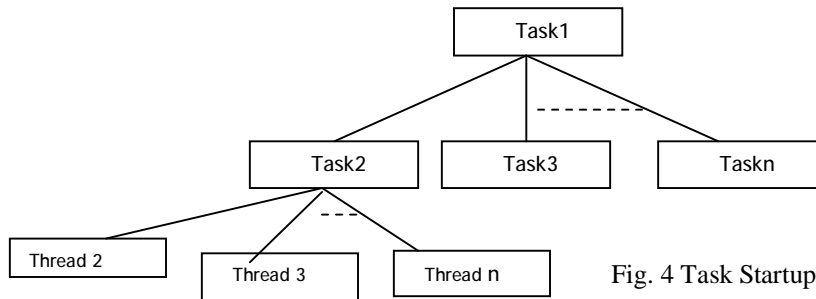


Fig. 4 Task Startup Tree

```

#define MAX_PROCS 20      /*total processors on the chip*/
void main()
{
    int procArray[MAX_PROCS] = {0,0,0,5,6,6,6,6,2,2,2,5,3,3,3,7,1,1,4,4}; /* application number
loaded on proc */
    int my_peid;
    my_peid = (_QUAD_INDEX * 4) + _PE_INDEX;
    if(my_peid == 0)
    {
        StartPrograms(procArray);      /* start 1 instance of each task*/
        start_other_pes(0, 2, procArray); /* other instances of this task */
    }
    my_peid = get_PE_number(); /* get my PE ID*/
    /*Perform useful computation*/
    terminate_program(my_peid, (Load_Table *)load_table_address, FREE_PL | FREE_PD); /*
terminate the task*/
}
StartPrograms(int *procArray)
{
    int i;
    for(i = 1; i < MAX_PROCS; i++) /* for all the tasks*/
    {
        if(procArray[i] != -1)
        {
            if(procArray[i-1] != procArray[i]) /* start only one instance of this task*/
                start_program_on_proc(procArray[i], i, 0);
        }
    }
}

```

Table 1: Code generated by the Mapping tool for one of the synthetic test application

The above code is from the main task launched on QUAD 0 PE 0.

During application modeling one specifies the buffer length to be used in the communication between different modules. This buffer length is used to allocate the shared memory structure. A shared memory structure consists of the following fields, Memory ID, Semaphore, Thread Count, Buffer Size, and Buffer Pointer.

Based on the tasks involved in communication the buffer will be allocated in local memory or shared DRAM. If all the tasks involved in communication are in the same cluster, then the shared memory is allocated in the local memory of the cluster. This would reduce the communication latency as the local memory access takes far less amount of time than DRAM. But, this is based on the availability of the local memory. If the tasks involved in communication are in different cluster, then the buffer is allocated in the Shared DRAM. Some communication buffers require to be guarded by a semaphore. Again, based on the tasks involved in communication, the semaphores are allocated from the local or global pool. The thread count indicates the number of threads involved in communication. The buffer can be freed when the thread count goes to zero.

For shared memory communication one has to generate structures to be used with the shared memory library. We developed this library, and it has a similar structure to that of the windows shared memory library API. The shared memory library structures

are generated in common header files. Once this code is generated, the user can integrate it with the core Intellectual Property modules developed. In the case of MAVD, the audio and video library forms the IP libraries.

This code is compiled and linked using the LD script generated to produce the executable. This executable is run on the hardware or the simulator to verify the results.

4 MAVD Modeling

The two main types of modeling in MILAN are Application Modeling and Resource Modeling. During application modeling, one builds the dataflow model in the dataflow aspect window. There are three main types of dataflow classes; Primitive, Compound, and Alternate. Compounds are composite dataflow nodes. They contain dataflow graphs themselves. For example, in MAVD, the Audio Decoder is a composite model. It will contain the PE component and the DSE component, which forms the primitive model. Alternates are similar to composite models, but they represent alternative implementation. Suppose a component can be executed on PE and on DSE then one would specify it as an alternate. During mapping one can select the processor based on the availability. For system instantiation only one alternate is selected. Figure 5 shows the MAVD model created in GME using MILAN paradigm. On the right hand window the folder structure is shown. MAVD is the root composite folder with different modules in it. These folders in the root folder are composite module and form root for the module. The communication between different modules are bi-directional and is shown using connection between in-port and out-port of the specific modules.

For each of the composite module there are primitive or composite modules within these modules. Figure 6 shows the Video Decoder Module. There are 6 Video Decoder threads running on 6 PEs and 12 Video Decoder threads running on 12 DSEs. The 6 PE threads are shown as 6 primitive components, while the 12 DSE threads are enclosed within a composite component, which has 12 primitive components. This makes design easy to understand.

In dataflow model there are two ways of showing shared memory communication. One way is to show it through connection between all the modules. This becomes complicated with too many connections. The other way is add a new primitive called SharedMemory in the window, which will show the shared memory communication between all the modules shown in the current window. We have defined a new Atom [13] called SSharedVariable in the MILAN paradigm.

Once a primitive component is defined, it needs to be mapped onto one type of processor. But before we map onto processor, we need to model these processors. This is done using the Resource Modeling paradigm in MILAN. MILAN provides support for resource modeling to model various architecture capabilities that can be exploited to perform design space exploration and drive a set of widely used energy and latency simulators from a single model. The Resource Modeling provides support for modeling

of various architectural components such as CPU, BUS, Memory, IO. One can even specify the type of cache used in the architecture. For each component one can specify what is the throughput, energy estimate, latency estimate and various other parameters. To specify that the current primitive module is going to run on a particular processor, one has to provide a reference to the processor in the primitive module in the mapping aspect.

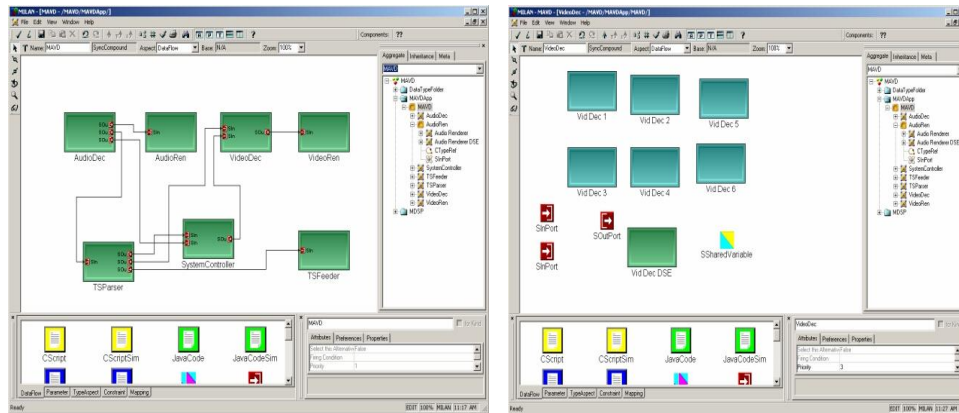


Fig. 5 Modeling MAVD using GME

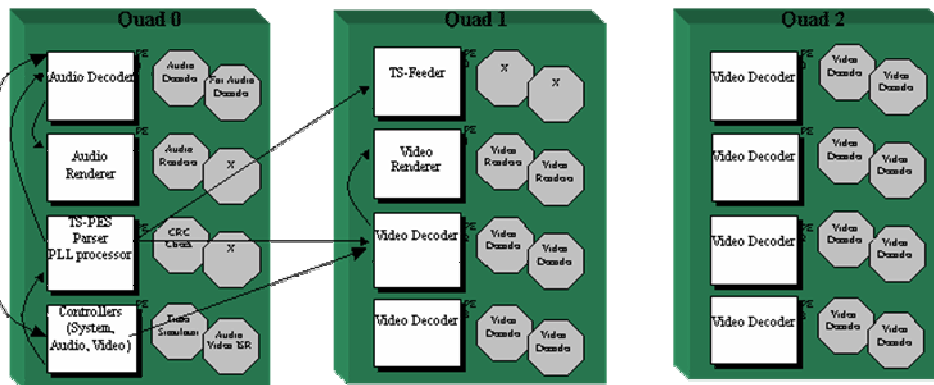


Fig. 6 MAVD Solution for MDSP Chip

Figure 6 shows the final mappin. In the MAVD system, the Video Decoder and Video Renderer has more communication, so they are placed within the same cluster. Similarly, Audio Decoder and Audio Renderer are placed in the same cluster. There is more communication between TS-PES Parser task and Audio Decoder task, so they have to be placed in the same cluster. Similarly, the System Controller has more

communication with Audio Decoder due to higher rater of Audio ISR than Video ISR and needs to be placed in the same cluster.

We compared the mapping obtained by our algorithm and the mapping done manually by experienced system designers from Cradle over the span of two years and found it to be same. The design had gone through multiple iterations before it was finalized, while we obtained it using our tool within the first iteration.

5 Related Work

One of the most popular hardware/software co-design tools is Ptolemy. It provides an environment for design, modeling, and simulation of concurrent heterogeneous embedded systems. Though Ptolemy provides different models of computation, the main problem with it is, it does not provide mechanisms for integration of external simulators.

Writing simulators of complex multiprocessor systems for the modeling environment is not a trivial task. Instead, one can use the existing simulators provided with these processors and use them in the modeling environment. This is precisely what MILAN offers. Easy integration of existing simulators, ease in modification of the paradigm to suit the underlying architecture.

There are other co-design or modeling projects for embedded system. Some of them are Polis [8], which uses finite state machine model for hardware/software co-design, Chinook [8], which focuses on IP composition, communication synthesis and rapid evaluation. In Chinook, one has to provide behavioral description of the application, target description and mapping function in order to synthesize the system. But none of these tools could suffice our need of simulator integration. The mechanism for mapping application onto the architecture was not present in Chinook. Instead, the user has to provide the allocation function. Though Polis provides mapping mechanism, it was not for clustered architecture like the one we are considering.

MILAN incorporates an optimal mapping algorithm for linear array of tasks [13]. But it requires the tasks to be executed in linear order, whereas in our case, the tasks are executed concurrently on different processors. Hence it cannot be used for our problem domain. There has been lot of research in scheduling theory for various types of problems [17] [18][19][20]. The solutions from these researches could not be directly applied to the problem under consideration.

6 Conclusions

In this paper we have presented tools to model complex high-performance embedded systems. We have validated our techniques by modeling a complex application using a

multi-core processor. The same techniques can be used to model complex applications on multi-chip boards.

The tools still require man-in-loop process to provide feedback to the modeling environment after simulation, based on the profiling results. The future direction of the research would be to automate this process, so that given a system, the tools would optimize the design automatically without much human intervention.

7 References

1. Arm Corporation, www.arm.com
2. Cradle Technologies, www.cradle.com
3. Azul Systems, www.azulsystems.com
4. Kestral Multiprocessor, <http://www.soe.ucsc.edu/research/kestrel/>
5. Orion Multisystems, www.orionmulti.com
6. ClearSpeed, www.clearspeed.com
7. Davis J. et al. , Overview of the Ptolemy project, ERL Technical Report CB/ERL No. M99/37, Dept EECS, University of California Berkeley, July 1999
8. Balarin F. et. al. : Hardware-Software Co-Design of embedded Systems: The POLIS Approach, *Kluwer Academic Publisher* 1997
9. Chou P. et. al.: IPChinook: An integrated IP-based design framework for Distributed Embedded Systems, *DAC* 1999
10. SystemC user Guide, <http://www.systemc.org>
11. "MILAN: A Model Based Integrated Simulation Framework for Design of Embedded Systems", Agrawal A., Bakshi A., Davis J., Eames B., Ledeczi A., Mohanty S., Mathur V., Neema S., Nordstrom G., Prasanna V., Raghavendra, C., Singh M. *Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES 2001)*, Snowbird, Utah, June 2001
12. "3400 Hardware Architecture Reference", Cradle Technologies, www.cradle.com.
13. Milan Manual, <http://milan.usc.edu>
14. Ledeczi A., et.al., The Generic Modeling Environment, *Workshop on Intelligent Signal Processing* , Budapest, Hungary, May 17, 2001
15. A. Nambiar and V. Chaudhary, "Mapping Resource Constrained Applications on Chip Multiprocessors", *Proceedings of the 2005 International Conference on Embedded Systems & Applications (ESA'05)*, Las Vegas, Nevada, June 2005.
16. E. G. Coffman, M. R. Garey, D. S. Johnson "Approximation algorithm for Bin Packing: A Survey, Appears in Approximation Algorithms for NP-Hard Problems", D. Hochbaum(ed) PWS Publishing(1996)
17. V. Chaudhary and J. K. Aggarwal, "A generalized scheme for mapping parallel algorithms", in *IEEE Trans. on Parallel and Distributed Systems*, Mar '93, pp. 328 - 346.
18. D. G. Feitelson and L. Rudolph, "Distributed hierarchical control for parallel processing." *Computer* 23(5), pp. 65-77, May 1990
19. Dror G. Feitelson, "Packing Schemes for Gang Scheduling", *Job Scheduling Strategies for Parallel Processing -- Proceedings of the IPPS'96 Workshop* (1996)
20. Y.-K. Kwok, I. Ahmad. "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors." *IEEE Trans. on Parallel and Distributed Systems*, vol.7, pages 506--521, March 1996