A Multiprogrammed Workload Model for Energy and Performance Estimation of Adaptive Chip-Multiprocessors

Mladen Nikitovic and Mats Brorsson

Department of Microelectronics and Information Technology,
Royal Institute of Technology
Electrum 229, SE-164 40 Kista, Sweden
mladen@imit.kth.se, matsbror@imit.kth.se

Abstract

Today, there is a trend towards steadily increasing functionality in mobile terminals. This trend in turn increases the performance demand on the architecture that is supposed to do all the work. It is likely that more traditional architectures like multi-processors are used in future mobile terminals. They are attractive because they can now be integrated on a single chip and can provide the desired performance efficiently if intelligently managed. Choosing the most efficient architecture configuration is however a complex issue and depends on multiple factors.

We believe that the way the behavior of the workload is modeled is of paramount importance when estimating the efficiency of any proposed architecture for future mobile terminals. Therefore, a deterministic and simple workload description is needed. In this paper, we show how such a multiprogrammed workload is created and used for energy and performance estimation of an adaptive chip-multi-processor (CMP) architecture.

1 Introduction

The never ending demand for faster, lighter, and cheaper mobile terminals such as cell phones and Personal Digital Assistants (PDAs) has increased the complexity of their computing platform. From initially consisting of simple RISC processors, they have evolved to superscalar processors. However, the increased performance provided by a new computing platform does not come for free. More complex processors consume more power and we cannot simply rely on the next generation of process technology to lower the voltage level in order to keep power consumption on a low level. Furthermore, superscalar architectures has performance limitations and it is simply a matter of time until more complex architectures are needed.

Parallel architectures, especially multi-processors, has the potential to satisfy the performance demand of the future. The advances of process technology has made it possible to design a chip-multiprocessor (CMP), thus a multi-processor system implemented on a single silicon die. The CMP architecture like other complex architectures can increase both power consumption and energy consumption if no measures is taken to reduce it. The benefit of a CMP is that it utilizes parallelism at several levels, thus both at instruction-level (ILP) and thread-level (TLP), whereas the uniprocessor only relies on available ILP for high performance, which is limited. To increase the performance of an uniprocessor one can increase the running frequency. The drawback is that once the frequency is increased, the voltage level also has to be increased to provide enough current to drive the transistors at the new frequency level. This way, the power consumption is increased quadratically. In a CMP on the other hand, one can add a processor to increase the performance, which only increases the power consumption linearly, since the frequency and voltage level is preserved. However, care still has to be taken to minimize the overall power consumption.

In this paper, we describe how simulation of an adaptive CMP architecture is done using a multiprogrammed workload model. Using this model, one can estimate both performance and power consumption of the CMP under realistic scenarios. To our knowledge, this is the first time a simple methodology for generating a realistic multiprogrammed workload model is proposed for modelling the behaviour of mobile terminals. The adaptiveness of the CMP comes from it's ability to activate and de-activate individual processors dynamically according to a predefined strategy, this to save as much power and energy as possible. The simulator is based on an extension of SimpleScalar, a microarchitectural uniprocessor simulator [3]. In our research studies we apply a bursty multipro-

grammed workload model, which we believe represents a typical behaviour pattern of mobile terminal applications.

The rest of the paper is divided into following sections. The next section describes related work. Section 3 describes the simulated architecture. Section 4 shows the simulation platform with its process scheduler, and performance and energy models. The multiprogrammed workload model is discussed in section 5. In section 6 we show our methodology for simulation and section 7 shows an example study using that methodology. In the final section we make some conclusions and discuss possible future directions of our work.

2 Related work

Modeling multiprogrammed workloads is a non-trivial issue and is today often overly simplified when performing studies of complex parallel hardware. Traffic models of e.g web, disk, and I/O systems on the other hand are quite well studied and has some characteristics that is desirable when modeling multiprogrammed workloads.

We believe that the challenge in modelling workloads of future mobile terminals is to model their unpredictable behavior i.e their burstyness. To our knowledge, very little work has been done to develop realistic multiprogrammed workload models with a bursty behavior. Previously proposed multiprogrammed workloads have typically been created by releasing a set of applications simultaneously without any considerations on the realism of their approach. Calzarozza et. al has made several important studies that show the different possible approaches to workload modelling such as queuing models, ON-OFF model, and self-similar traffic models [4].

The ON-OFF model, also known as train model, was first introduced in [8] to model packet arrival patterns in a token ring networks. A train consists of a sequence of “on” and “off” periods modelled by Pareto distributions. During ON periods, packets are arriving and during OFF periods no packets arrive at all.

There are studies indicating that a traffic on a network is an aggregation of the bursty traffic generated by independent sources, which means that the burstyness increases with the number of sources. It has been previously discovered that the burstyness has self-similar characteristics [17]. In this paper, we use a traffic model that has bursty, self-similar characteristics and map it to a multiprogrammed workload.

The design-space of a CMP is very complex and it is not trivial to determine which configuration is most efficient. Usually, designers do not have the resources nor the time to wait for a chip to be manufactured before evaluation can be done. Therefore, many resort to simulation as a way of evaluating the efficiency of a design in a reasonable

amount of time. Simulation can be done on different levels of granularity. At the lowest level, a hardware description of a design can accurately be evaluated using existing hardware simulation tools. The drawback is that these tools are very slow due to their high accuracy. At the highest level of granularity, the desired architecture design is simulated using simple models. The benefit is that simulation results can be generated quickly. This is done at the expense of accuracy. There are therefore a range of different type of simulators and we will go through some of them.

To simulate and analyse a multi-processor system, one could resort to different types of simulators. Full-system simulators has the potential to simulate an entire system, thus a multi-processor system with operating system (OS) and I/O devices. This could be useful, because the OS can have a noticable impact on the effectiveness of an application execution. SimOS is a freely available full-system simulator that runs on Unix multi-processors such as SGI Challenge [15]. It simulates a multi-processor system that is similar to the hardware of the host. Therefore, it can run on the real hardware using direct-execution mode to speed up simulation. The drawback of direct-execution mode is that relevant statistics is harder to gather. Using the other option, which is to use binary translation, reduces the effectiveness; to simulate a multi-processor system with four processors and caches results in very high slowdown. SimOS simulates a Irix operating system on top of the simulated hardware, which makes the simulated hardware together with the OS a combination that does not represent a common case in most modern mobile systems.

Simics is another full-system simulator can run on various hosts and simulate different target architectures, multi-processor systems included [11]. This simulator has been commercialised, thus it is not freely available. The multi-processor capabilities of Simics closely resembles the multi-processor support of SimOS. Bochs is a freely available full-system emulator of x86 platforms with limited multi-processor support [1].

None of the full-system simulators considered to estimate the power consumption of various system components, which is important in studies related to embedded and mobile systems. Some of their multi-processor simulation support is limited and the multi-processor architecture does not always represent an architecture suitable for mobile terminals.

There are also a number of multi-processor simulators available [7, 12, 18, 10]. Most of them simulate only the user-mode execution of an application. Among them, the Rice Simulator is one of the most popular [7]. RSIM simulates a multi-processor system where each node consists of a detailed MIPS R10000 model, and a two-level cache hierarchy. The nodes are connected through a two-dimensional mesh network. ML-RSIM is an extension of RSIM, which include a OS and device models [16]. Today, ML-

Figure 1: An example CMP

RSIM is currently incapable of simulating a multi-processor system, thus it simulates only uni-processor systems.

3 The target architecture

The proposed multiprogrammed workload model can be applied to both uni-processor and multi-processor environments but we are targeting CMP because we believe that this architecture is promising in terms of delivering performance in an efficient manner. The simulator simulates a symmetric shared memory multi-processor architecture. Each processor in the system is a simple single-pipelined low-power core with private instruction and data caches. The processors are connected through a shared atomic bus. Also, there is a unified level 2 cache, which is shared among the processors. Coherency between the caches is maintained using the MESI coherence protocol. The processors, L1 caches, L2 cache, and the shared bus is assumed to be on-chip, whereas the DRAM is located off-chip. Figure 1 shows an example of such a CMP with four processors.

Modern embedded processors usually has the possibility to run in different power-saving modes. We will assume that the processors in our CMP is capable to run in three different modes: running, standby, and dormant. In running mode, the processor is simply executing instructions, whereas in standby mode the processors clock-tree is disabled, which saves dynamic power consumption. In the final mode, dormant mode, the processor and cache is electrically disabled. The standby mode only reduces the dynamic power consumption, whereas the dormant mode saves both static and dynamic power consumption. Table 1 shows the different stated power modes and the perform-

Table 1. Processor power modes.

Mode	Description	Recovery Time
RUN	Processor executing instructions.	≥ 1 cycle
STANDBY	Clock-tree disabled.	≥ 10 cycles
DORMANT	Power rails off.	≥ 100 cycles

Figure 2: Abstraction of simulation platform

ance overhead associated with each mode when returning to running mode.

4 Simulation platform

The aim of the simulation platform is to simulate the targeted CMP architecture. In our case, this has been done by extending an existing simulator to be capable to simulate a CMP architecture. We started with sim-cache, a functional cache simulator from the SimpleScalar simulation suite [3] and extended it so that simulation of the target architecture is possible. The processors are simulated functionally. An interconnect module with arbitration mechanism was added between the level 1 and level 2 caches to provide the means for communication between processors.

The simulation platform can be seen as an entity consisting of multiple layers (see Figure 2). At the highest layer, there is a simple process scheduler with power-saving capabilities, thus it provides the adaptiveness of the CMP. The next layer consists of the timing models of each component of the CMP, thus the processors, caches, interconnects, and the off-chip bus and DRAM. Finally, the lowest layer estimates the power consumption of each corresponding sub-component in the system. Each of these layers of the simulation platform will be more closely described in the following sections.

4.1 Process scheduler

The adaptiveness of the CMP architecture comes from its ability to dynamically decide whether a processor should be active or not using the process scheduler. The expected service provided by a process scheduler is to allocate processes to processors according to an algorithm and to perform taskswitching. The role of the process scheduling algorithm is to decide which of the ready processes to select for execution. In our case, we use the Earliest Deadline First (EDF) process scheduling algorithm.

In addition to process scheduling, our scheduler also decides which processor is going to be activate and in-activate according to a predefined power-saving strategy. The reason why the power-saving strategies are implemented in the process scheduler is because it has access to the information regarding the current load on each processor at any time instant. We have also added the possibility to use

dynamic voltage scaling (DVS) in uni-processor mode, which is a good comparison to our proposed techniques. In a previous study [13], we measured the efficiency of the two scheduling strategies described below. Each strategy will affect two important events 1) the way process allocation is done and 2) how processes are migrated.

The first strategy, S1, put a higher priority on low response time while the second strategy, S2, puts a higher priority on low power consumption while still meeting deadlines. Both strategies impacts the scheduler's way of doing process allocation and process migration. The scheduler will in both strategies put idle processors into dormant mode. The following pseudocode describes the scheduler's process allocation and migration policy with respect to selected strategy. The following pseudocode shows the differences between the two strategies.

Process allocation pseudocode using S1

```
if(awake & idle CPU available)
    allocate task to that CPU
else if(dormant CPU available)
    wake up CPU and allocate task to it
else
    allocate task to CPU with least load
```

Process migration pseudocode using S1

```
if(task gets highest priority on new CPU)
    migrate task
else
    do nothing
```

Process allocation pseudocode using S2

```
if(task w/out deadline {
    if(active CPU available)
        allocate to active CPU w least load
    else
        allocate task to dormant CPU
}
else {
    if(CPU w only non-critical tasks available)
        allocate task to that CPU
    else if(active CPU available)
        allocate to least loaded CPU
    else
        wake up sleeping CPU and allocate task to it
}
```

Process migration pseudocode using S2

```
if(two CPUs only have ready non-critical tasks)
    move ready tasks to least loaded CPU and shut down CPU w empty ready queue
else
    do nothing
```

The idea behind S1 strategy is to provide a good quality of service regardless whether the process is time-critical or not. The second strategy puts a higher priority on time-critical processes by trying to execute them on processors with low load where they would get higher priority. Strategy S2 minimizes static power consumption by waking up a dormant processor as a last resort. This way, as many processors as possible are put into low-leakage mode.

This is just an example of how a low-power strategy integrated with the process scheduler could be used to save power. The strategies could of course be varied in an infinite number of ways.

4.2 Timing models

The performance of the target architecture is estimated using several timing models. Each of the sub-components in the target architecture has a corresponding timing model. The timing of the processor model is simple, each execution of an instruction takes one cycle if no cache miss is experienced. This timing model is suitable when simulating a single-pipelined processors, where the maximum executed instructions per cycle (IPC) is 1. However, if a cache miss is experienced, then the delay associated with fetching the data or instruction from the next level of the memory hierarchy is added to the total execution time.

The interconnect is modelled as a shared atomic bus. The arbitration unit chooses who is going to be master of the bus at any given time instant. In this case, the arbitration is done in a FIFO manner, where the processor with the lowest ID number has the highest priority in case there are multiple requests from different processors at the same time. The atomicity of the bus guarantees sequential consistency and that a memory operation on the second level cache or the main memory must be completed before the next bus transaction can take place. The timing models of the caches are the same as the models originally used in the SimpleScalar sim-outorder simulator. The original main memory timing model in the simulator was updated to reflect the timing of a more modern DDR DRAM chip running at 133 Mhz.

4.3 Energy models

The power and energy consumption of each sub-component in the system is in most cases estimated using capacitance models. In remaining cases, as in the processor energy model, a simple crude model based on technical documents figures is used. The power consumption of each processor in the system is estimated using the data from published figures [5]. Power is consumed only when an instruction is executed. At other times, such as when the processor is waiting for a memory operation to complete, dynamic power in the processor is assumed to be minimized through the use of clock gating. The energy models of the caches are the same as the ones used in the Wattch

Figure 3: A sample trace using the b-model.

simulator [2]. The power and energy consumption of the interconnect is estimated using wire capacitance models [14]. The energy model of the off-chip bus and memory is estimated using models from the IRAM project [6].

Using these models, one can get a good estimate of the total dynamic power consumption of the CMP. However, static power consumption is projected to be an equally important factor to consider in the future. Today, the static power consumption is contributing about 10% of the total power consumed in a CMOS device. We therefore include static power consumption in our estimations due to its increasing importance. This is done by measuring the time spent in running mode or standby mode, where static power is still consumed, and multiply that time with 10% of the respective sub-components maximum power consumption. This way, we can get an estimate of how much of the total power consumption does come from static power consumption.

5 Workload model

Today, it is not uncommon that the multiprogrammed workload model used when evaluating a parallel platform is quite simple in its design. Usually, evaluation is done using a few applications executed simultaneously. However, it is expected that the workload of future mobile terminals is underdeterministic in its behaviour, thus it is not sufficient to base a model around a scenario where one releases all applications at the same time. Today, the load changes from long periods of idleness to sudden high-demanding peaks. We believe that the load posed on a mobile terminal is highly dependent on the behaviour of the user, thus it has the potential to make the workload very bursty due to user requests. Our approach is therefore to use a workload model originally developed for bursty traffic modelling [19] and map it to a multiprogrammed workload.

The traffic model, b-model, is based on the observation that most network, I/O, and web traffic has a self-similar bursty pattern. Previously proposed models using poisson arrival models cannot fully re-create a self-similar nor bursty traffic behaviour. The b-model is closely related to the “80/20% law” in databases, which means that 80% of

Figure 4: Random side-selection b-model.

the queries involve 20% of the data, thus parameter b represents a ‘bias’, a b equal to 0.8 means that at a given time interval, 80% of the accesses happen on one half of the time interval and the remaining 20% on the other half, thus a b equal to 0.5 would create a load that is evenly distributed. The trace can be scaled horizontally by specifying an aggregation level (n) that is larger than 1, thus an aggregation level (n) of 10 would distribute the traffic over 2^{10} time intervals. To scale the traffic vertically one can increase the size of the data set (N), thus increasing the data set N from 10 to 100 would increase the number of data points with a factor of 10.

Figure 3 shows a sample model using data set (N) 4096, aggregation level (n) 10, and b equal to 0.7. Here, the subdivision is done so in such a way that b is always ends up on the left side of a subdivision, which can be seen by looking at the “slopes” of the peaks that always lean towards the right side in the trace. One could create randomness in the trace generation by randomly putting b on either right or left side. Figure 4 shows the a trace with same parameters as in Figure 3, but now generated with randomly putting b on either right or left side at each subdivision point.

In our research studies, we are interested in modeling a multiprogrammed workload with a bursty behaviour. Each data point in a trace generated using the b-model corresponds to a uniform operation e.g. reading 1 KB from a disk. As stated earlier, we are mostly interested in the underdeterministic behavior provided by the b-model. When a trace is created, we make a number of transformations. First, we transform the height of each peak to a discrete value. This is done by passing the remainder of the current data point to the next time instant. This way, retain the total number of data points is retained in the trace. In the second transformation, we choose to replace each data request with a release of an application. The end-result is a multiprogrammed workload with bursts of application releases.

5.1 Modeling multiprogrammed workloads

The benefit with this traffic model is that it models the bursty behavior that we believe comes from user interaction with the terminal. Using this model, we translate each

data point to an arrival time of an application. Of course, one could interpret the number of data points at a given time instant in the graph as the current workload level. This approach makes the mapping from the data point to a release of an application much more complex. Also, this interpretation results in a discrepancy between the given data set N and the number of executed applications in the multi-programmed workload.

We need applications to make the mapping between the traffic model and the multiprogrammed workload model possible. Since we are interested in mobile terminals, we selected our applications from a benchmark suite that consists of applications typical in telecommunications environments [9]. Figure 5 shows the workload after each data discrete data point has been replaced with a release of an application using a model with parameter b equal to 0.45, aggregation level (n) 10, and data set (N) 256. This workload model consists of 256 application executions and it can be seen from the figure that the highest workload peak consists of 12 applications running at the same time. This workload would for example force the process scheduler to time-share between the applications on a CMP with less than 12 processors, a scenario that is not always tested.

6 Simulation methodology

We use simulation as a methodology to evaluate the efficiency of our proposed platform. Our simulator, sim-acmp, is similar to any simulator from the SimpleScalar suite, because it uses the same code base as the original simulators in that suite. The difference can be seen in Figure 6 where the inputs and outputs of the simulator are specified. When using a simulator from the SimpleScalar suite, it is common to use a configuration file which specifies certain parameters of the target architecture. Also, only a single process or application is simulated at a time which is specified on the command line. In our case, we want to simulate multiple processes at the same time, thus we need a way to specify them. This is done by passing a workload file to the modified simulator that loads the specified processes into the memory. This required modification in the original process loader module.

Figure 5: A heavy multi-programmed workload.

Figure 6: Simulator inputs and outputs.

The output of the simulator is partially modified and extended. The original simulator, sim-cache, was functional i.e it did not provide any timing statistics. We extended the simulator with our timing models so that the statistical output would include timing characteristics. Also, since we are interested in the power consumption we extended the statistical gathering of data with information about the consumed power and energy. More information about the timing and energy models can be found in section 4.2 and 4.3. We will now explain more closely how the workload file is specified.

6.1 Workload file specification

Our multiprogrammed workload scenarios are constructed off-line, thus all applications released during a workload scenario are specified in a workload file that is later used by the simulator. Each line in the workload file is specified according to the following format specified by Table 2.

Table 2 Workload specification fields.

Field name	Meaning of field
Application type	Application characteristics.
Release time	Release time of application.
Period	Periodicity if periodic process.
Deadline	Deadline of time-critical process.
CPUlist	List of allocateable CPUs.
Handle	Process name.
Tracefile	Path to tracefile.

Each specific field is more closely explained below.

Application type. Execution of both aperiodic and periodic applications are supported in our simulator. An aperiodic application is specified with application type ‘A’ and a periodic application is specified with application type ‘P’. If the release time of the application is non-zero, then one can add a ‘R’ specifier to specify that the release of this application is going to occur at an instant in time specified by the next field. If the application has a deadline, then the ‘D’ specifier is added to specify that a deadline is associated with that application. The supported combinations of application type specifiers are PRD, ARD, PD, AD, PR, AR, P, and A.

Release time. This field is only specified if the ‘R’ specifier is used in the application type field, otherwise this field is left empty. An application can only be released on an instant in time that is a multiple of the OS period. The default OS period is 20ms, thus a release time of 15 would correspond to a period time of 300ms.

Period. The duration of a period is only specified if the application type specifier includes an ‘P’ character, otherwise it is empty. In that case, the value of the period is specified in the same as the previous field, the release time, which is a multiple of the OS period. Thus a period of 2 means that the application will be released every 2nd OS period, thus each 40th ms if OS period is 20ms.

Deadline. The deadline is only specified if the ‘D’ specifier is included in the application type, otherwise it is empty. The deadline is specified in absolute time in seconds, thus a deadline of 0.03 would mean that the deadline is 30ms.

CPUlist. The CPUlist contains a list of processor IDs that the application is allowed to execute on. The list always begins with a leading ‘[’ and ends with a ‘]’. Between these specifiers, there is a list (each ID is separated with a comma) that specifies with processor IDs are allowed to execute this particular application. The first processor in the CMP has ID 0, second has ID 1, and so on.

Handle. The handle field is a name chosen by the user to easily refer to the application when the OS-scheduler outputs scheduling statistics.

Tracefile. Each application in the workload is generated as a trace, using sim-eio. The tracefile field specifies the absolute path to the trace file that is supposed to run when an application is released.

After the workload file has been read, the corresponding task control block (TCB) is created for each application and the simulation begins. A processor that has nothing to do will execute an idle process if no particular power-saving scheduler strategy is used.

At the end of the simulation, the simulator outputs statistics in the same fashion as any other SimpleScalar version of sim-cache would do. There are couple of statistics added to this version, since it simulates both timing and energy consumption of the architecture.

7 Example of a study

In a previous study [13], we have measured the efficiency of the scheduling strategies described in section 4.1. We did that using two workload scenarios, a light workload consisting of 64 application executions (see Figure 7) and a heavy workload consisting of 256 application executions (see Figure 5). Our proposed CMP is compared against a comparable uni-processor with DVS. All architecture configurations are assumed to be capable of aggressive clock-

Figure 7: A light workload scenario.

gating when the a component is idle. In our case, we use cc3 clock-gating style which was provided by Wattch. When using the cc3 clock-gating style, one assumes that an idle component is immediately disabled, thus minimizing the dynamic power consumption.

Figure 8 shows the dynamic and static energy consumed by architectures we considered during heavy and light workload scenarios (H/L prefix), thus uni-processor with and without DVS, 4-way CMP without scheduling strategies, 4-way CMP with S1 or S2 strategy. The bars are normalized against the 4-way CMP without any strategy. Using DVS on the uni-processor only decreased the energy consumed during the light workload. We can see that the 4-way CMP without any strategy consumes as much as a uni-processor with DVS during a light workload and several times less energy during the heavy workload. Using S1 strategy, the 4-way CMP reduced its energy consumption by 78% during the light workload. Energy consumption was further reduced using S2 strategy by 8% at the expense of response-time of non-critical applications while deadlines of time-critical applications are still met.

8 Conclusion

In this paper, we have shown how a multiprogrammed workload mode can be used to estimate the performance and energy consumption of a CMP architecture. Starting from an existing traffic model that can be generated in a single pass with a single parameter (b), we have mapped it to a multiprogrammed workload with a bursty behavior, a

Figure 8: Estimated energy consumption.

behavior we expect from future mobile terminals. This model is used in conjunction with our simulator that includes energy and timing models of each sub-component in the system. An example study shows how the efficiency of adaptive scheduler strategies can be evaluated using the multiprogrammed model.

In the next phase of our workload modelling effort, we would like to correlate our generated workloads with different real-world workloads. It is possible that future mobile terminals not only approaches the complexity of today's workstation architectures, but also their usage pattern. If there is a similarity, one could make comparisons of workstation workloads against our multi-programmed workloads.

Acknowledgements

The research in this paper is partly supported by the Swedish Foundation for Strategic Research within the INTELECT and SoCWare programmes.

References

- [1] Bochs documentation: <http://bochs.sourceforge.net/>
- [2] D. Brooks et. al., Wattch: A Framework for Architectural-level Power Analysis and Optimizations. In Proceedings of the 27th International Symposium on Computer Architecture, pp. 83-94, 2000.
- [3] D. Burger, T. Austin. The SimpleScalar Tool Set, Version 2.0. Technical Report CS-TR-97-1342, University of Wisconsin, June 1997.
- [4] M. Calzarossa, L. Massari, and D. Tessa, Workload Characterization - Issues and Methodologies. In G. Haring, C. Lindemann and M. Reiser, editors, Performance Evaluation - Origins and Directions, volume 1769 of Lecture Notes in Computer Science, pp. 459-484, Springer-Verlag, 2000.
- [5] L. T. Clark et. al., An Embedded 32-b Microprocessor Core for Low-Power and High-Performance Applications, IEEE Solid-State Circuits, Volume 36, Issue 11, pp. 1599 - 1608.
- [6] R. Fromm et al., The Energy Efficiency Of Iram Architectures. In Proceedings of IEEE International Symposium on Computer Architecture, pp. 327-337, 1997.
- [7] C. J. Hughes, V. S. Pai, S. V. Adve, Rsim: Simulating Shared-Memory multiprocessors with ILP Processors, In IEEE Computer, pp. 40-49, 2002.
- [8] R. Jain and S.A. Routhier. Packet Trains - Measurements and a New Model for Computer Network Traffic. In IEEE Journal on Selected Areas in Communications, SAC-4(6), pp. 986-995, 1994.
- [9] C. Lee et. al., Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. In Proceedings of International Symposium on Micro architecture, pp. 330-335, 1997.
- [10] D. Magdic, Limes: a multiprocessor simulation environment for PC platforms, In Proceedings of International Conference on Microelectronics, Volume: 2, pp. 841-844, 14-17 Sept. 1997.
- [11] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, B. Werner, Simics: A Full System Simulation Platform, In IEEE Computer, pp. 50-58, 2002.
- [12] S.S. Mukherjee, S.K. Reinhardt, B. Falsafi, M. Litzkow, M.D. Hill, D.A. Wood, S. Huss-Lederman, J.R. Larus, Wisconsin Wind Tunnel II: a fast, portable parallel architecture simulator, In IEEE Concurrency, Volume: 8 Issue: 4, pp. 12-20, Oct.-Dec. 2000.
- [13] M. Nikitovic and M. Brorsson. A Low Power Strategy for Future Mobile Terminals. In Proceedings of the International Conference on Design, Automation, and Test in Europe (DATE). To appear as an interactive presentation. Feb. 2004.
- [14] J.M. Rabaey and M. Pedram, Low Power Design Methodologies. 1996, ISBN 0-7923-9630-8.
- [15] M. Rosenblum, S.A Herrod, E. Witchel, A. Gupta, Complete Computer System Simulation: The SimOS approach, In IEEE Journal of Parallel and Distributed Technology, pp. 34-43, 1995.
- [16] L. Schaelicke, M. Parker, ML-RSIM Reference Manual, tech. report 02-10, Department of Computer Science and Engineering, Univ. of Notre Dame, Notre Dame, Ind., 2002.
- [17] J.F. Shoch and J.A. Hupp. Measured Performance of an Ethernet Local Network. In Communications of the ACM, 23(12), pp. 771-721, 1980.
- [18] M. Wakabayashi, H. Amano, Environment for multiprocessor simulator development, In Proceedings of International Symposium on Parallel Architectures, Algorithms and Networks, pp. 64-71, 7-9 Dec. 2000.
- [19] M. Wang, T. Madhyastha, N. H. Chan, S. Papadimitriou, C. Faloutsos, Data Mining Meets Performance Evaluation: Fast Algorithms for Modeling Bursty Traffic, In IEEE Proceedings of International Conference on Data Engineering, pp. 2001.