Empirical Macromodeling of Operating System Communication Primitives

Mirko Loghi
Dipartimento di Informatica
University of Verona
Strada Le Grazie, 15
37134 Verona, Italy
loghi@sci.univr.it

Massimo Poncino
Dipartimento di Informatica
University of Verona
Strada Le Grazie, 15
37134 Verona, Italy
massimo.poncino@univr.it

Luca Benini
DEIS
University of Bologna
Viale Risorgimento, 2
40134 Bologna, Italy
lbenini@deis.unibo.it

ABSTRACT

The capability of estimating the energy consumption of software in multi-processor systems-on-chip (MPSoCs) is crucial for enabling quick evaluation of both software and hardware optimizations. However, true high-level power estimation should be applicable at the software level, possibly through effective power models depending on parameters which can be derived directly from the characteristics of the applications.
In this work we propose an energy model for the communication primitives which, in spite of its simplicity, allows to model the traffic-dependent nature of energy consumption by means of an abstract parameter, namely, the size of the message exchanged during communication.
Preliminary results show that the model has an average error below 5%.

1. Introduction

Multiprocessor systems-on-chip (MPSoCs) are becoming the most natural platform for running embedded applications. They are commonly used in multimedia and mobile devices and they are expected to become even more widespread in the future. Two are the distinctive features characterizing MPSoCs.
First, they are constrained by severe power budgets [1], since they are often battery-powered. Furthermore, the power dissipation strongly affects reliability, because it depends on the average working temperature. Therefore, there is an increasing need for solutions that reduce energy consumption, which requires significant efforts in every phase of the design flow, from technology to software development. Clearly, increasing the abstraction level when searching for energy-efficient solutions can provide higher savings, yet it poses significant challenges for what concerns the ability of predicting the impact of these optimizations, that is, power estimation.
The second characteristic of MPSoCs is their "communication-centric" nature. The scaling of technology towards deep submicrometric devices tends to increase the relative importance of interconnect delays. This fact, together with the difficulty in developing MPSoCs into a single clock domain, is causing a shift in the design paradigm of SoC towards the so-called "Networks on Chip" (NoC). Communication between processors will in fact become similar to that of conventional *computer* networks, possibly with non-deterministic communication between processors. From the application point of view, communication tasks are then earning a central role in the MPSoC scenario, and systems are becoming more and more "communication-dominated".
Under such a scenario, the availability of high-level *power models for the communication primitives* has become an essential infrastructure for driving the choice of possible optimizations. Such models should rely only on high level parameters, available during the application development, or on factors that can be obtained from fast simulation of the application. In fact, even though in principle software power estimation can always be performed by running the application on a full-system power estimator, this approach is too slow to be useful in the inner loop of power optimization and design space exploration.
In this work we derive an empirical macromodel which relates the energy consumption of communication primitives to the characteristics of the applications running on the system. Intuitively, the energy required by a communication primitive (e.g. sending a message) is a function of the size of the unit to be exchanged and of the traffic on the shared medium. The former determines the intrinsic effort required for moving the data, whereas the latter accounts for the non-deterministic interference between the activity of the various processors.
Unfortunately, while the "message size" can be easily inferred from the application source code, "traffic" is poorly quantifiable at the application level. The model proposed in this work removes the dependency on the traffic variable, and reduces the model to a very abstract dependency between energy and message size.
For our analysis we leverage on a multiprocessor simulation platform, fully equipped with power models for the main system components. The simulator allows to simulate realistic applications as well as an underlying operating system, specifically based on message-based interprocess communication. It is precisely these primitives that we try to model.
Preliminary results show that the proposed macromodel can be successfully used for the intended purposes.

2. Related Work

Software power estimation and power modeling have been actively studied in recent years. However, most of the research has been focused onto single-processor systems. The seminal work by Tiwari et al. [2] introduced the popular *instruction-level power analysis* approach, which builds a model associating power consumption to instructions or instruction pairs, based on a set of characterization experiments.
Better accuracy and, above all, better resolution (at the price of increased execution time), can be achieved by *micro-architectural power models* [3, 4], which rely on the knowledge of the main functional units of a microprocessor (e.g., execution units, register file, etc.).
All the above-mentioned approaches focus on the CPU, but software execution consumes power also in the memory system, system buses, and peripherals. For this reason, software power estimation must account for all system components. Several researchers [5, 6, 7, 8] proposed *full-system estimators*, that couple instruction set simulators with CPU, memory, bus and peripherals power models.

To avoid full instruction-level simulation, several techniques have been proposed for characterizing software power consumption at a level of granularity much coarser than the single instruction. Macro-modeling techniques have been proposed for sub-routine calls [9] within an application program, for operating system calls [10, 11, 12], and even for entire tasks [13, 14]. The main advantage of these techniques is that they can provide early feedback on the power consumed by complex programs, in presence of significant middleware support, without the computational cost of a detailed instruction-level simulation.

A completely different class of approaches is based on an abstract representation of the multiprocessor system, in terms, for instance, of a queue network or a Petri net, where processors are modeled as requestors, and buses and memories as queues or places [15, 16, 17, 18]. These approaches provide analytical performance models which are in principle applicable at a very high-level of abstraction, provided that the suitable parameters can be extracted from the inspection of the application.

In this sense, our macromodel is closer in scope to these latter models in that it is meant for use at the application level. However, it differs in several aspects. First, it models energy for a well-defined level of granularity, namely the communication primitives of the operating system. Second, it is an empirical model. As such, it requires characterization on a specified target architecture using pre-validated energy models for the components of the system. Third, it is based on a single parameter which can be extracted in a straightforward way from the analysis of the application.

3. Multiprocessor Platform

The simulation platform used in this work is composed of (i) a configurable number of 32-bit ARM processors, (ii) their private memories, (iii) a shared memory, (iv) a hardware interrupt module, (v) a hardware semaphore module, (vi) the 32-bit ST Microelectronics STBus interconnect, used to connect all the above modules. The interrupt device allows processors to send interrupt signals to each other, while the semaphore device implements *test-and-set* operations. Both these devices are mapped in the addressing space and are used for interprocessor communication and synchronization purpose.

The system is configurable in several of its components, such as the memory latencies, the number and the size of the caches, the topology of the bus, and many others. For further details, the reader is referred to [19]; Figure 1 shows a conceptual block diagram of the system architecture.

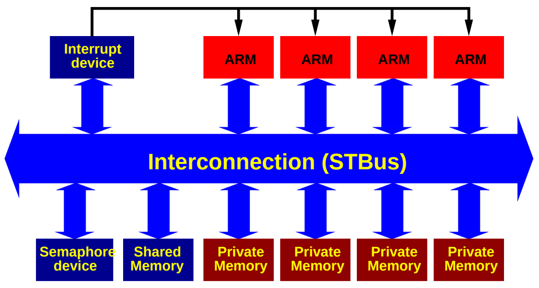

Figure 1: High-Level Architecture of the Platform.

For the experimental part of this work we used a four-CPU system, with 8 KB instruction cache and 4 KB data cache, memories with a two-cycle latency, and a shared bus as the interconnection topology.

The MPARM platform has power models for its components. The power models are technologically homogeneous, as they are obtained from foundry data related to the same technology (STMicroelectronics 0.13 μm). Since the simulations are cycle-accurate and the power models are invoked at each cycle, MPARM can, on a cycle-by-cycle basis, provide how much energy is spent by any of the various component (core, cache(s), memories and bus).

The platform also includes a complete operating system and its support APIs. RTEMS [20] is a light-weight OS suitable for embedded systems, it offers complete support for multiprocessing, and provides a complete API for inter-processor communication and synchronization. Applications can directly use the communication primitives provided by the API to implement parallel programs. The target of this work is exactly the characterization of these communication primitives.

4. Characterizing Communication Primitives

4.1 Macromodeling

Our target is to determine the energetic cost of the communication operations performed by an application and, by aggregation, the energy spent by the whole application in its communication tasks. Generally speaking, the energy required by an operation consists of the product of two terms: a capacitive factor modeling the intrinsic cost of the operation and the time required for that operation. All capacitive factors can be considered technology constants, so that energy models actually reduce in this context to timing models.

Intuitively, the energy cost is a function of the size of the unit to be exchanged (the *message size*, hereafter), and of *the traffic* in the system. The former quantity determines the intrinsic effort required for moving the data around, whereas the latter account for the non-deterministic interference between the activity of the various processors. However, while the message size can be easily inferred from the application source code, traffic is a dynamic quantity. Furthermore, the two factors are not independent: traffic depends in fact on the message size, since larger messages imply higher amount of traffic.

In a message passing environment, such as the one considered here, characterizing communication primitives amounts to characterize the sending and the reception of a message. For simplicity, let us focus on the "send" operation, with reference to the target architecture of Figure 1.

Sending a message implies some bus and memory accesses, each one requiring a variable amount of time. However, while the time required for a memory access can be regarded as constanti, depending only on technological parameters and on the platform specifications, the time spent for a bus access is strictly related to the instantaneous traffic on the (shared) medium. Therefore, the main difficulty to overcoming the estimation of the energy spent in the communication is to obtain an *estimation of the traffic on the medium.* Such traffic depends on the execution patterns of all the processors, not only the ones which initiates the data exchange, and this cannot be predicted without accurate simulations or, at least, some kind of analytical traffic models.

Our challenge here is to *model the communication cost as a simple function of the message size.* In other words, we try to hide the dynamic nature of the traffic inside a very high level parameter. The motivation is that to send a message, an application does many operations, as synchronization, memory allocation, memory validation and so on. All these operations are composed of a large number of assembly instructions, each one requiring a variable amount of time and energy which cannot easily be determined at a high level of abstraction. But the total energy consumption of a message exchange is the sum of all such contributions, and this will tend to mask the non-deterministic behavior of the individual terms.

So, it should be possible to obtain a good estimation of the time (and thus energy) spent by a message passing primitive of an operating system. As a result, it should be possible to obtain also a high level macromodel with an acceptable error (say, below 10%).
The main drawback of any macromodel approach is its strong dependency on the target platform. The model must require the characterization of the target architecture and, in case the latter changes, the characterization should be repeated from scratch. From the programmer perspective, however, this is not an issue, since the platform is usually completely defined when the application development is starting. Reconfigurable systems are an exception to this situation; however, they offer a limited degree of variability on the configurable features, so in principle it is possible to build several macromodels, to cover the whole range of variations.

4.2 RTEMS Interprocessor Communication

In this section we will analyze the communication primitives of the operating system running on our platform, in order to understand how to apply the above modeling principles. We will provide a high level description, in order to give an idea of the system's behavior during a data exchange.
In RTEMS, the basic application-level inter-thread communication primitives are *message queues*. Threads communicate by writing/reading *messages* into/from a queue, using two system calls: `rtems_message_queue_send` and `rtems_message_queue_receive` (`send` and `receive`, for short).
Assuming that the processor P_1 `send`s a message to processor P_2, the operation executed are:

1. P_1 obtains a pointer to a buffer in shared memory.
2. P_1 fills the buffer with the desired data using the `memcpy()` function of the standard C library.
3. Once the shared buffer has been manipulated, P_1 notifies (via an interrupt) to P_2 that it can proceed.
4. P_2 gets the address of the shared buffer.
5. The data are explicitly copied from this shared buffer to a local buffer (via `memcpy`).
6. P_2 sends an acknowledge response to P_1.

The `receive` system call works in a similar manner, although not fully symmetrically. The `memcpy` function is used to actually carry the data, while many other tasks have to be performed, as synchronization and the exchange of the address of the shared buffer.

4.3 Statistical analysis

The six macro-operations described in the previous section are actually complex operations, each one consisting of a large number of machine instructions. These instructions can be classified into two categories: instructions which access the bus and the memory (*remote instructions*), and instructions which can be completed using only the cache without requiring an access to the shared bus (*local instructions*). Under a zero-miss assumption in the cache, the first class consists only of *instructions which access shared objects*, which are not cached. In fact, all other instructions accessing the private memories will not need to access the bus, because the required data can always be found in the cache. Such assumption is justified by experimental data, that show hit rates higher than 98%, on average.
Based on this considerations, the energy spent for executing a local instruction can be considered as a constant, since it does not depend on the traffic. Conversely, the energy spent executing a *remote instruction* will depend on the traffic on the bus, which impacts the time and, as a consequence, the energy required from a processor to gain the access to the medium.
We can thus write an expression of the energy spent for a communication primitive (`send` or `receive`), as a function of traffic and message size, as follows:

$$E_{comm} = \sum_{i=1}^{N_0} E_i^{local} + \sum_{i=1}^{N_1(S)} E_i^{remote}(U_i) \tag{1}$$

Where N_0 is the number of local instructions, N_1 is the number of remote instructions and S is the message size. E_i^{local} is the energy spent for a local instruction, inclusive of the energy spent by the core and of the energy used for the cache access; E_i^{remote} is the energy spent for a remote instruction (taking into account the bus and the memory access also), and U_i is the bus utilization (i.e. the traffic) when executing the i-th instruction.
Equation 1 shows that E_i^{remote} is a function of the traffic on the bus, while E_i^{local} can be treated as a constant, with respect to S and U_i. Furthermore, N_0 is a constant, because it represents the number of instructions that perform "accessory" operations of the given primitive (e.g. preparing a message); conversely, N_1 strongly depends on the message size S, because it directly affects the inner loop of the `memcpy`. The dependency of N_1 on S can be written as:

$$N_1 = C_0 + C_1 \cdot S \tag{2}$$

The term C_0 models the fact that a fixed number of accesses to shared objects are needed for the synchronization and validation, while the actual data exchange requires a number of elementary bus transfers linearly dependent on the message size. In fact, for a data bus with of D bytes, the transfer of S bytes requires S/D elementary bus transactions.
Our purpose is now *to remove the dependence on U_i's*, obtaining a model which is only dependent on S. Abstracting the traffic variable from E_i^{remote} transform this quantity into a random variable, whose probability density can, in principle, be estimated by measuring it on a large number of operating conditions, and whose average and variance are respectively μ_{remote} and σ^2_{remote}. This operation yields:

$$E_{comm} = N_0 \cdot E^{local} + N_1(S) \cdot \mu_{remote} \tag{3}$$

In the first term we have emphasized the independency of E^{local} of S, while in the second term the value $E_i^{remote}(U_i)$ has been replaced by its average value.
Substituting now Equation 2 into Equation 3 we have:

$$E_{comm} = (N_0 \cdot E^{local} + C_0 \cdot \mu_{remote}) + C_1 \cdot \mu_{remote} \cdot S \tag{4}$$

from which the linear dependency of E_{comm} on S is exposed. The absolute error of this model is roughly $\sqrt{N_1} \cdot \sigma_{remote}$, which is the standard deviation of the second term in Equation 1. This value corresponds to the standard deviation of a sum of N_1 random variables, assuming statistical independence among the variables E_i^{remote}.
The relative error is the ratio of the absolute error and the target measure:

$$\varepsilon_r = \frac{\sqrt{N_1} \cdot \sigma_{remote}}{E_{comm}} \tag{5}$$

$$= \frac{\sqrt{N_1} \cdot \sigma_{remote}}{N_0 \cdot E^{local} + N_1(S) \cdot \mu_{remote}} \tag{6}$$

The expression shows that the relative error is small if N_0 is greater than N_1 (as it occurs in practice), and it decreases when N_1 increases.

4.4 Results

We have used Equation 4 as a model template, and computed the actual model as follows. First, we have run characterization tasks, in which we have measured the quantity under measure E_{comm} for many different operating conditions. For this purpose we have written a few synthetic benchmarks parameterized with respect to S, and relative to a four-processor configuration of the simulation platform. In these benchmarks one of the processors sends a message to another one, while the others two generate tunable dummy traffic on the bus.

For the various runs, energy values are collected so that a distribution of the energy consumption for different message sizes can be built. Figure 2 shows the plot of the distribution for `send`.

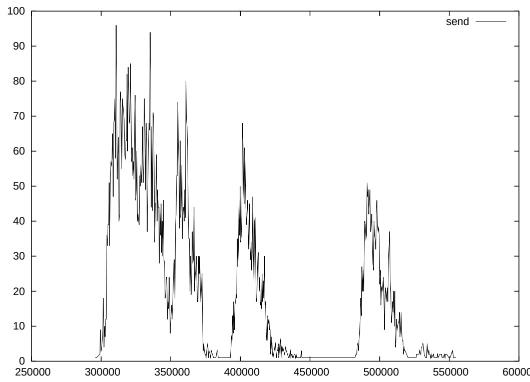

Figure 2: Distribution of Energy for `send`.

We observe a multi-modal distribution, with peaks related to the various message sizes used in the characterization. We also notice that, as expressed by Equation 5, the absolute error (the support of the sub-distributions) is small compared to the the average value, yielding a small relative error. For instance, the two rightmost peaks (larger energy values) have roughly the same support (the base of the peak), but different average values. This assesses the result that the error decreases for larger values of S (and thus of energy).

Finally, we have applied least-mean square regression to the set of collected points in the (E_{comm}, S) space to extract the actual model, getting an equation of the type

$$E_{comm} = \alpha + \beta \cdot S$$

as in Equation 4. Clearly, we have different models for `send`s and `receive`s.

Such model fits quite well the collected data, showing an intrinsic average error below 2%. Maximum errors are about 10% for the `send` and about 13% for the `receive`, but these worst-case errors occur very seldom, and the average error is the most typical case.

In order to evaluate the validity of the model, we ran several simulations, with message sizes different from the ones used in the characterization phase, and with a different traffic behavior. The resulting average errors are below 5% for both `send` and `receive` while the worst case errors are below 11% and 18% respectively.

5. Conclusions

In MPSoC, modeling of bus traffic is central for characterizing the time and energy cost of communication primitives. In this work we show that very good accuracy can be achieved by means of simple empirical macromodels which depend only on a very abstract parameter such as the size of exchanged messages.

Our analysis shows that the non-determinism of bus traffic can be hidden inside a higher-level parameter, with negligible loss in accuracy.

6. REFERENCES

[1] L. Benini, M. Poncino, "Ambient Intelligence: A Computational Platform Perspective" in: *Ambient Intelligence: Impact on Embedded System Design*, T. Basten, M. Geilen, H. de Groot eds. Kluwer Academic Publishers, 2003.

[2] V. Tiwari, S. Malik, A. Wolfe, "Power Analysis of Embedded Software: a First Step Towards Software Power Minimization," *IEEE Transactions on VLSI Systems*, Vol. 2, no. 4, pp. 437-445, Dec. 1994.

[3] D. Brooks et al., "Power-Aware Micro-Architecture: Design and Modeling Challenges for Next-Generation Microprocessors," *IEEE Micro*, Vol. 20, no. 6, pp. 24-44, Nov.-Dec. 2000.

[4] N. Vijaykrishnan et al."Evaluating Integrated Hardware-Software Optimizations using a Unified Energy Estimation Framework," *IEEE Transactions on Computers*, Vol. 52, no. 1, pp. 59-76, Jan. 2003.

[5] T. Simunic, L. Benini, G. De Micheli, "Energy-Efficient Design of Battery-Powered Embedded Systems," *IEEE Transactions on Very Large-Scale Integration Systems*, Vol. 9, no. 1, pp. 15–28, Feb 2001.

[6] S. Gurumurthi, et al. "Using Complete Machine Simulation for Software Power Estimation: the SoftWatt Approach," *International Conference on High-Performance Computer Architecture*, pp. 124-133, 2002.

[7] J. Henkel, Y. Li, "Avalanche: an Environment for Design Space Exploration and Optimization of Low-Power Embedded Systems," *IEEE Transactions on VLSI Systems*, Vol. 10, no. 4, pp. 454-468, Aug. 2002

[8] T. Givargis, F. Vahid. J. Henkel, "Instruction-Based System-Level Power Evaluation of System-on-a-Chip Peripheral Cores," *IEEE Transactions on VLSI Systems*, Vol. 10, no. 6, pp. 856-863, Dec. 2002.

[9] M. Lajolo, A. Raghunathan, S. Dey, L. Lavagno, "Cosimulation-Based Power Estimation for System-on-Chip Design," *IEEE Transactions on VLSI Systems*, Vol. 10, no. 3, pp. 253-266, June 2002.

[10] T. Tan, A. Raghunathan, N. Jha, "Embedded Operating System Energy Analysis and Macro-Modeling," *International Conference on Computer Design*, pp. 515-222, 2002.

[11] A. Acquaviva, L. Benini, A. Ricco', "Energy Characterization of Embedded Real-Time Operating Systems," in L. Benini, M. Kandemir, J. Ramanujam, *Compilers and Operating Systems for Low Power*, Kluwer Academic Publishers 2003.

[12] R. Dick, G. Lakshminarayana, A. Raghunathan, N. Jha, "Analysis of Power Dissipation in Embedded Systems using Real-Time Operating Systems," *IEEE Transactions on CAD*, Vol. 22, no. 5, pp. 615-627, May 2003.

[13] R. Dick, N. Jha, "MOGAC: a Multi-Objective Genetic Algorithm for Hardware-Software co-synthesis of distributed embedded systems," *IEEE Transactions on CAD*, Vol. 17, no. 10, pp. 920-935, Oct. 1998.

[14] A. Acquaviva, E. Lattanzi, A. Bogliolo, L. Benini, "A Simulation Model for Streaming Applications over a Power Manageable Wireless Link," *European Simulation and Modeling Conference*, Oct. 2003.

[15] "A Multiprocessor Bus Design Model Validated by System Measurement", T.-F. Tsuei, M.K. Vernon, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 6, November 1992, pp. 712–727.

[16] "System-level Performance Analysis for Designing On-Chip Communication Architectures," K. Lahiri, A. Raghunathan, S. Dey, *IEEE Transactions on CAD*, Vol. 20, No. 6, June 2001, pp. 768–783.

[17] "SPI - A System Model for Heterogeneously Specified Embedded Systems," D. Ziegenbein, K. Richter, R. Ernst, L. Thiele, J. Teich, *IEEE Transactions on VLSI Systems*, Vol. 10, No. 4, August 2002, pp. 379–389.

[18] "A Formal Approach to MpSoC Performance Verification," K. Richter, M. Jersak, R. Ernst, *IEEE Computer*, Vol. 36, No. 4, April 2003, pp. 60–67.

[19] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, R. Zafalon, "Analyzing On-chip communication in a MPSoC Environment", *DATE'04: Design, Automation and Test in Europe*, Feb. 2004, pp. 752–757.

[20] RTEMS home page, `www.rtems.com`.