Dottorato di ricerca in Ingegneria dell'Informazione
Politecnico di Milano - Dipartimento di Elettronica e Informazione

Proposal submission form for the doctoral thesis research project - First year
(Schema del programma di ricerca per la tesi di dottorato - Primo anno)

Thesis title:

C-LANGUAGE SOURCE-LEVEL POWER CONSUMPTION ESTIMATION AND OPTIMIZATION FOR EMBEDDED SYSTEMS

Titolo della tesi:

STIMA E OTTIMIZZAZIONE DEL CONSUMO ENERGETICO PER SISTEMI EMBEDDED A LIVELLO DI CODICE SORGENTE IN LINGUAGGIO C

Ph. D. Student: Daniele Paolo Scarpazza
Ph. D. Cycle: XVIII
Thesis advisor: Prof. William Fornaciari

Abstract

Thanks to the wide diffusion of personal communication, computing and entertainment devices, the market of portable, battery-powered, embedded systems is quick expanding, emphasizing the importance of energy consumption estimation and optimization techniques.

Current circuit-level and gate-level estimation techniques are accurate, but computationally extremely demanding, and too fine-grained to provide results useful for the system-level design.

In the mid of 90's, a new research thread was born, now coming to maturity, whose goal was to achieve energy consumption estimation at the instruction-level: estimation flows based upon these methodologies are now mature products, which provide accurate and reliable results.

The objective of our research is to step on towards the next abstraction level, more precisely, we desire to achieve methodologies to estimate the energy consumption at the source code level (e.g., C language source code); moreover, we desire to implement an estimation flow which is accurate and usable, capable of applying those methodologies semi-atomatically.

Previous works on the impact of source-level transformations on power consumption, proved that gains attainable at source level largely outmatch those attainable at the lower levels; in fact source level optimization can rely on a deeper knowledge of the semantics of the project and can operate on both algorithms and data structures. In the light of these considerations, we are interested in the construction of an optimization flow able to assess the impact of source-code transformations known in literature onto energy consumption.

Riassunto

Con la grande diffusione dei dispositivi personali di comunicazione, calcolo e riproduzione multimediale, il mercato dei sistemi embedded portatili (e quindi alimentati a batteria) sta subendo una forte espansione, portando in primo piano l'importanza delle tecniche di stima e ottimizzazione del consumo di potenza.

Le tecniche di stima della potenza circuitali e gate-level sono accurate, ma computazionalmente molto onerose, e a grana troppo fine per poter fornire indicazioni utili alla progettazione a livello di sistema.

Dalla metà degli anni novanta si è aperto, e sta giungendo a maturazione, un filone di ricerca che si proponeva di effettuare stime di consumo energetico a livello di istruzione: flussi di stima e ottimizzazione basati su queste ricerche sono ormai prodotti consolidati, che offrono risultati precisi e accurati.

L'obiettivo della nostra ricerca è compiere il passo successivo, che porti al livello di astrazione immediatamente superiore, in particolare ci proponiamo di ricercare metodologie di stima del consumo energetico a livello di codice sorgente (per esempio in linguaggio C); ci proponiamo inoltre di realizzare un

flusso di stima accurato e usabile, in grado di applicare tali metodologie in modo semiautomatico.

Lavori precedenti, riguardanti l'impatto delle trasformazioni a livello di codice sorgente sul consumo di potenza, hanno messo in evidenza che i guadagni ottenibili a quel livello surclassano quelli ottenibili ai livelli inferiori, potendo contare su di una conoscenza più approfondita della semantica del progetto, e potendo operare contemporaneamente su algoritmi e dati. Alla luce di ciò, desideriamo realizzare un flusso di ottimizzazione che valuti l'impatto sul consumo di potenza delle trasformazioni di codice sorgente note in letteratura.

1 Area of Investigation

This Ph. D. thesis lies in the area of investigation of embedded systems low-power design methods. It also indirectly involves the following research areas:

- source level optimizations;
- formal languages;
- low-power architectures;
- optimizing compiler design;
- power measurement electronics.

It involves, among many others, the following industrial sectors:

- semiconductor industries;
- electronic design automation tool vendors;
- intellectual property (IP) block vendors;
- telecommunication equipment manufacturers;
- consumer electronics manufacturers;
- automotive industries;
- space industries;
- ...

2 Research objectives

Power dissipation is increasingly becoming a limiting factor in the integration of complex SoC (System-on-Chip) designs. This is starting to impact the mobility of ubiquitous computation and communication, as well as affecting the cost and reliability of communication systems and networks. Therefore, recent design trends of embedded applications require the analysis and optimization of performance and power in all the components of the system, especially during the first stages of the design, when alternative solutions are compared. The current pervasiveness of microprocessor-based architectures is enforcing the importance of a fast analysis of the software, carried out through the entire development / compilation flow, namely from the source level down to the assembly.

To this purpose, in literature several authors explored the problem to determine bounds on the execution time of a process running on a microprocessor; unfortunately, this task is becoming more and more complicated with the current CPU features. Many investigations attacked the problem at a coarse grain, focusing mainly on the WCET (Worst Case Execution Time), because the target application is often real-time constrained. In general, the problem has two aspects: (1) program paths analysis, to determine which sequences of instructions will be executed and how they influence the WCET; (2) microarchitectural analysis, i.e. the modeling of the hardware system executing the program. Efficient estimation of WCET is, however, a hard task: to be decidible it requires absence of recursive functions, function pointers and bounded loops and, in addition, the scope of the path information strongly impacts the estimation accuracy. In any case, the analysis is computationally hard and thus not suitable during design space exploration, when frequently the proper hw/sw architecture has to be chosen by quickly comparing alternatives. Microarchitectural-related issues, modeling the timing analysis of the assembly instructions, are another source of possible estimation errors.

Our goal is to provide a framework, general enough to gather a broad range of specification formalisms, as well as to take into account the peculiarities of the actual processor on which the software will be finally mapped. In particular, we focused on the problem of efficiently estimating the average timing features of the code, making at the same time the formulation open to easily incorporate a statistic characterization of the parameters, if necessary. It constitutes the base for hw/sw partitioning or simply to analyze the impact of different microprocessors on the performance. Work is in progress to extend it towards the analysis of the software power at the source level.

2.1 An historical perspective

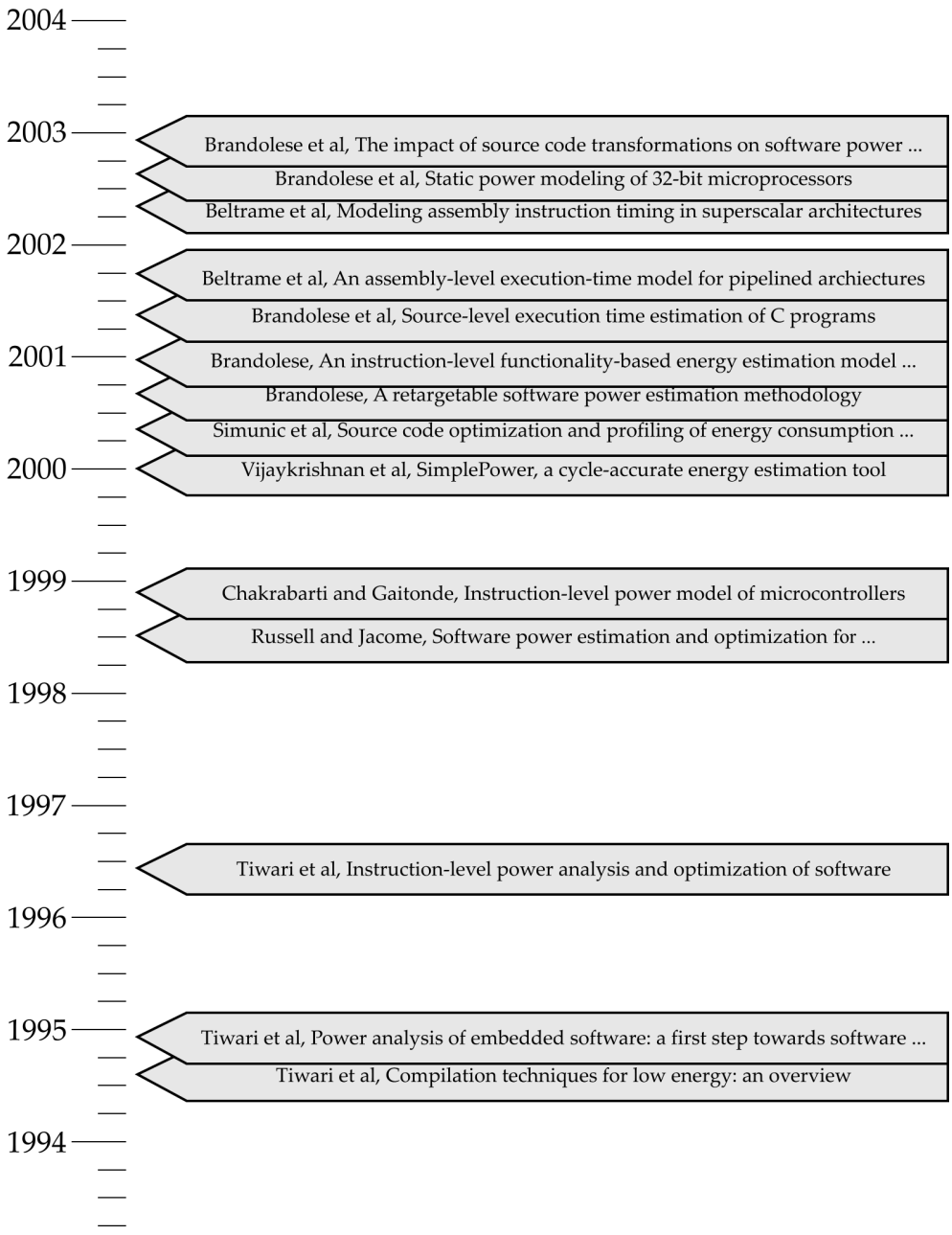

Figure 1: Time-line representation of the major works in this research field.

3 State of the art

In this section we briefly examine the most important works in the field of software power optimization. A time-line representation is given in Figure 1.

In mid 90's, it was already clear that software power consumption was a critical parameter in the design of embedded systems (though for reasons which are slightly different from current ones) and that traditional attempts to model that power consumption, based on the detailed physical layout of the processor, were inadequate for a number of reasons: often layout information of a CPU is not available to the application designer; additionally, estimation techniques based on layout are expensive, difficult to apply and computationally very demanding.

Tiwari et al. [1], in 1994, were the first recognize the need of an instruction-level model of power consumption: they set up an experiment based on a Intel development board, with a 486DX2 processor and current probes on board; then they executed long loops made of the same instruction repeated a large number of times, in order to fill all the pipeline stages with the same instruction, thus making it possible to eliminate any transient effects and to measure the average current absorbed by that instruction. Tiwari et al. were able to derive a power model based on instruction base costs, inter-instruction effects and influence of stalls and cache misses.

Two years later, the same authors provided an overview of software energy optimisation techniques [1], showing the possibility of beneficial impact on power of the optimization of techniques such as register allocation to reduce memory accesses, energy-driven instruction selection during compilation, instruction reordering for to reduce switching activity, software-controlled power management, and others.

Though the expectations of some of the above optimization techniques were destined to be harshly lowered by successive investigations [4], the importance and usefulness of instruction-level power estimation continued to grow, with the expansion of battery-powered devices.

In year 2000 a cycle-accurate simulator called *SimplePower* appeared [5] (based on a popular architecture simulator called SimpleScalar), which made it possible to estimate energy consumption of the CPU, memory system and buses, thus opening the way to easy evaluation of the effect of high-level algorithmic, architectural and compilation trade-offs.

A number of similar approaches to cycle-accurate power estimation was adopted in [6, 7, 9, 10, 12, 13]: authors developed a model based on functional units, instruction queues and resources, losing any physical or electrical detail, and reducing them to a simple parameter: the average functional unit current per instruction. Within this framework, they were able to model a number of pipelined processors, to simulate execution on them of arbitrary code, and to accurately estimate the number of stalls for each instruction, thus leading to a precise indication of consumed energy.

Motivated by the above results, in the last few years, the idea that it was possible and convenient to estimate power consumption at a source code level

appeared [15], and that source-level transformations could have a much higher impact on power optimization [16, 17] than assembly-level optimizations. In fact, compile-time optimizations have a partial perspective on algorithms and no possibility to introduce significant modifications to data structures, whereas source-code transformations can exploit full knowledge of the algorithm, modifying data structures and algorithms at the same time.

3.1 Shortcomings of the above solutions

Software power estimation techniques operating at different levels of abstraction have been proposed. Most of the approaches analyse the code at assembly level relying on an RT- or behavioural-level model of the microprocessor. These methods suffer two major drawbacks: the need of an accurate description of the microprocessor core and extremely long simulation run times. Recently, these problems have been partially overcome by focusing on an abstract (functional) model of the processor [13], but no reliable and widely applicable strategies working at source-level have been proposed in literature and no commercial or prototype tools are currently available.

Concerning the power optimisation, many proposals are focused on reducing the power consumed by memories, trying to minimize the need to access main memory through a careful management of data organisation, while exploiting the presence of caches. Other techniques operate at the instruction-level, and are based on a proper modelling of the microprocessor to provide power figures for each instruction. Power optimisation is then accomplished by selecting the best mix of instructions to execute the given application. A few techniques, still working on the code, for specific classes of applications can reduce power demand by decreasing the accuracy of computation within the limit of tolerability. Dynamic variable-voltage techniques have been also proposed, to reduce performance and power consumption, during the execution of weakly critical sub-tasks. Dynamic power management can be also employed at system level, by means of a suitable methodology forcing (sleep) low power states, when the system becomes inactive; both fixed and predictive strategies have been proposed. Hence, the task of managing the software-specific power-issues is not yet shifted toward EDA vendors. In particular, there is a lack of methodologies and tools working at source level both for estimation and optimisation of power consumption.

4 Research methodology

4.1 Theoretical models

Our research relies on a theoretical model of the C language (a model which could in principle be applied to any imperative language), first introduced in [15]. This model tries to capture in a general manner the structure of a program, model the language in a constructive and hierarchical. In order to do so, it

defines an 'elementary component', called *atom*. A critical point in defining the atoms of a language is the choice of the granularity at which the language is analyzed. To this purpose it is helpful to describe the language, in this case C, using formal grammars.

We introduce an incomplete grammar, where the set of terminal symbols comprises the usual terminals `=`, `(`, `)`, {, }, `for`, `return`, etc, plus the additional symbols *var*, *expr* and *cond*. Upon this terminal symbols it is possible to define all the expressions, statements and constructs of the C language. The outcome is an incomplete grammar (not listed here for sake of brevity) that lacks productions for all the symbols assumed as terminals, but it is useful to formally define the concept of atom. An atom is each nonterminal of the grammar, and it is composed by the collection of all the terminal only symbols in the right-hand side of any of its productions. Non-terminals appearing in the right-hand side of productions are not considered part of the atom but independent atoms themselves, though related to each other in a hierarchical fashion. The atom tree obtained by representing such an encapsulation relation is a simplification of the parsetree.

Thanks to a careful definition of the above grammar, it is possible to assign to each atom a (time, energy, size) cost which depends only on the semantics of the terminal symbols contained in the atom.

The source code of the project under estimation can be instrumented in such a way that, running the project, it is possible to determine the exact atom trace, and consequently the number of executions of any single atom.

To afford the complexity of the problem it is useful to consider the ideal conditions defined by these assumptions:

- the target architecture has an unlimited number registers and all the variables of the code are allocated to a fixed register;
- the initial value of the variables is pre-loaded in the corresponding register and thus no explicit initialization is required;
- inter-atom compiler optimizations are neglected;
- intra-atom compiler optimizations are neglected.

The first two bullets of this list lead to an underestimate of the real execution time while the last two tend to produce an overestimate. Preliminary tests show that the above effects tend to compensate to zero.

Under this simplifying assumptions, the determination of the execution time and consumed energy of the project run consists in a simple summation of the number of times each atom was executed over the cost in time and energy of each atom. The costs of executing each atom must be determined in advance; this can be done in a number of ways and at different levels of accuracy; a deeper tractation of this topic is beyond the scope of this paper.

The model includes parameters that allow to statistically fine-tune the estimators in order to compensate the above simplicistic assumptions and second-order effects.

4.2 Experimental developments

In this section we describe the desired structure of the source-level estimation flow, called *e3tools*, which is the major software artifact resulting from this thesis work. This flow is currently in the development stage of *working prototype*.

Given a generic project written in C language, the source-level estimation flow allows to divide that source code listings in their constituting atoms and to consistently instrument, compile and trace the project. The trace is then analyzed, thus obtaining the contribution of each atom, line of code and C function to the overall consumption of time, energy and code size. Figure 2 depicts all the constituents of the `e3tools` flow.

The role of each of the blocks in Figure 2 is described here:

- (1): C-language source files constituting the project to be analyzed;
- (2): header files, part of the project to be analyzed;
- (3): compatibility include-file, specialized on the actual compiler used in phase (14);
- (4), `pre`: compatibility preprocessor, internally relying upon GNU `cpp`; it preprocesses each input source code file which constitute the project, resolving possible implementation-dependent symbols;
- (5): preprocessed C-language source file; this file is free from file inclusions, macro definitions and other preprocessor directives;
- (6), `democritos`: a C-language parser which recognizes and characterizes atoms in the input source files;
- (7): atom coordinates and associated syntactic information, in a format which is exemplified in the following excerpt:

```
BEGIN FUNCTION main
[ (35,0) #231
‘  [ (47,2) #57
‘  ‘  ( (47,2) #56 OPERATOR () // function call
‘  ‘  ‘  < (47,2) #53 IDENTIFIER lame_init
‘  ‘  ‘  > (47,10)
‘  ‘  ) (47,15)
‘  ‘  { (48,2) #69
‘  ‘  ‘  ( (48,5) #60 OPERATOR ==
‘  ‘  ‘  ‘  < (48,5) #58 IDENTIFIER argc
‘  ‘  ‘  ‘  > (48,8)
‘  ‘  ‘  ‘  < (48,11) #59 CONSTANT 1
‘  ‘  ‘  ‘  > (48,11)
‘  ‘  ‘  ) (48,11)
‘  ‘  ‘  [ (48,14) #68
‘  ‘  ‘  ‘  ( (48,14) #67 OPERATOR () // function call
‘  ‘  ‘  ‘  ‘  < (48,14) #61 IDENTIFIER lame_usage
‘  ‘  ‘  ‘  ‘  > (48,23)
‘  ‘  ‘  ‘  ) (48,36)
‘  ‘  ‘  ] (48,37)
‘  ‘  } (48,37)
                    ...
‘  ‘  [ (137,2) #228
‘  ‘  ‘  ( (137,2) #227 OPERATOR () // function call
```

```
‘  ‘  ‘  ‘  < (137,2) #224 IDENTIFIER lame_mp3_tags
‘  ‘  ‘  ‘  > (137,14)
‘  ‘  ‘  ) (137,19)
‘  ‘  ] (137,20)
‘  ‘  { (138,2) #230
‘  ‘  ‘  < (138,9) #229 CONSTANT 0
‘  ‘  ‘  > (138,9)
‘  ‘  } (138,10)
‘  ] (138,10)
‘  IMPLICIT RETURN #232
] (139,0)
END FUNCTION main
```

The above format indicates beginning and ending coordinates of each syntactic element of interests, expressed as a (line,column) couple. The flavour of parentheses used indicate the instrumentation style required for each syntactic elements (see (12)), numbers preceded by “#” signs indicate atom identifiers;

- (8): atom characterization information, in the following format: atom identifier, line number, atom characterization, function to which it belongs. The above format is exemplified in the following excerpt:

```
53    47    Null                main
56    47    FunctionCall        main
60    48    IntRelation         main
67    48    FunctionCall        main
75    56    FunctionCall        main
78    58    RegStructDot        main
79    58    IntNot              main
84    60    FunctionCall        main
85    60    IntNot              main
88    70    MemAssign           main
96    72    FunctionCall        main
97    72    MemAssign           main
```

- (9): list of function calls, in the following format: original file name (as opposed to preprocessed file name), line number (both in original and preprocessed source files, please note that the preprocessing process conserve line numbers), line number again (for compatibility reasons with the GUI), called function prototype. The above format is exemplified in the following excerpt:

```
main.c   60   60  extern int     strcmp   (const char * __s1, const char * __s2 )
main.c   72   72  extern FILE *  fopen    (const char * __filename, const char * __modes )
main.c   73   73  extern int     fprintf  (FILE * __stream, const char * __format )
main.c   74   74  extern void    exit     (int __status )
main.c  126  126  extern int     fwrite   (const void * __ptr, int __size, int __n, FILE * __s )
main.c  122  122  extern int     fprintf  (FILE * __stream, const char * __format )
main.c  123  123  extern void    exit     (int __status )
main.c  127  127  extern int     fprintf  (FILE * __stream, const char * __format )
main.c  128  128  extern void    exit     (int __status )
```

- (10): list of function definitions, in the following format: original file name (as opposed to preprocessed file name), beginning line number (both in original and preprocessed source files, same considerations as above apply), ending line number again, defined function prototype. The above format is exemplified in the following excerpt:

```
main.c 34 139 int  main (int argc, char * * argv )
```

- (11): list of function declarations, in the following format: original file name (as opposed to preprocessed file name), beginning line number (both in original and preprocessed source files, same considerations as above apply), ending line number again, defined function prototype. The above format is exemplified in the following excerpt:

```
/usr/include/stdlib.h   142  142  extern double   atof  (const char * __nptr )
/usr/include/stdlib.h   144  144  extern int      atoi  (const char * __nptr )
/usr/include/stdlib.h   146  146  extern long int atol  (const char * __nptr )
```

- (12), `stradivari`: an instrumentation tool; `stradivari` generates a new version of the C source file where each subexpression, statement and function body is instrumented in order to emit, namely;
  - expressions are instrumented by enclosing them in plain parentheses and separating the tracing code with a comma, thus forming a *comma expression*, in the following way: expression `expr` becomes `( tracing_call, expr )` ;
  - statements are instrumented by enclosing them in curly parentheses and separating the tracing code with semicolons delimiters, in the following way: statement `statement;` becomes `{ tracing_call; statement; }` ;
  - function bodies are instrumented by enclosing them in curly parentheses and adding tracing code both at the beginning and at the end, in the following way: expression `{...}` becomes `{ tracing_call; {...} tracing_call; }` ;

- (13): the instrumented version of the source file;

- (14): the GNU `gcc` compiler. Please note that the portion of the flow including steps (3) thru (12) was designed to externally behave as close as possible to the behavior of `gcc`; consequently, it was possible to automate the workings of this part of the flow by creating a script called `gcc`, referred to as *fake gcc* from now on, which is invoked in place of the real `gcc` and internally performs steps (3)-(12);

- (15): object files;

- (16): an object file containing the implementation of the tracing function;

- (17): the fake `gcc` script is called to perform linking; it actually calls the real `gcc`, adding the tracing function object;

- (18): the instrumented, executable form of the project. This executable behaves exactly as the original, non-instrumented form, with the exception that each atom execution causes the addition to trace file (18) of a log entry.

- (19): the execution trace;
- (20): the mapping of atom characterizations over instructions of a given architecture;
- (21): the cost of each instruction in terms of execution time, energy consumption and object code size;
- (22), `taylor`: accumulates the costs of each atom found in the trace, thus determining aggregate cost statistics per line of code and function;
- (23): a set of output report files, one for each source file at step (1), containing at each line the estimated execution time, consumed energy and occupied size of the code contained in the same line of the corresponding file (1);
- (24): an annotated dynamic function call graph, where for each node the amount of time, energy and size is indicated;

A more detailed description of this flow can be found in [18].

4.3 Data collection

Data to be used both for model tuning and model validation are mainly energy consumption data. Energy consumption data can be obtained in two main ways:

- **by measurement**: in many cases it is convenient to assume the supply voltage as a constant and to actually measure the absorbed current and the execution time only; in the case of pipelined CPUs, appropriate countermeasures have to be applied to avoid data biasing due to secondary effects;
- **by simulation**: the possibility to actually perform power measurements depends on the availability of measured of the system under measurement and of appropriate measurement equipment, none of which are assured at the time of the writing of this paper. Additionally, in a number of cases it is important to perform software evaluation on theoretical architecture models, for which the silicon has not yet been manufactured.

4.4 Criteria for evaluation of results

After model tuning and internal validation, for any developed model, a battery of external validation tests will be performed. We expect to achieve good values of quality-of-result indicators, such as high (> 0.95) coefficients of correlation between real and estimated data, low relative prediction errors and possibly gaussian approximated error density.

4.5 Assessment of results by end users

All of the industrial partners involved in POET, the research project to which this work belongs, which are individually described in section 6.1, have strong interests in the availability, quality and usability of the software products developed in the context of this research. It will be their role to provide a hands-on, field evaluation and feedback on our results.

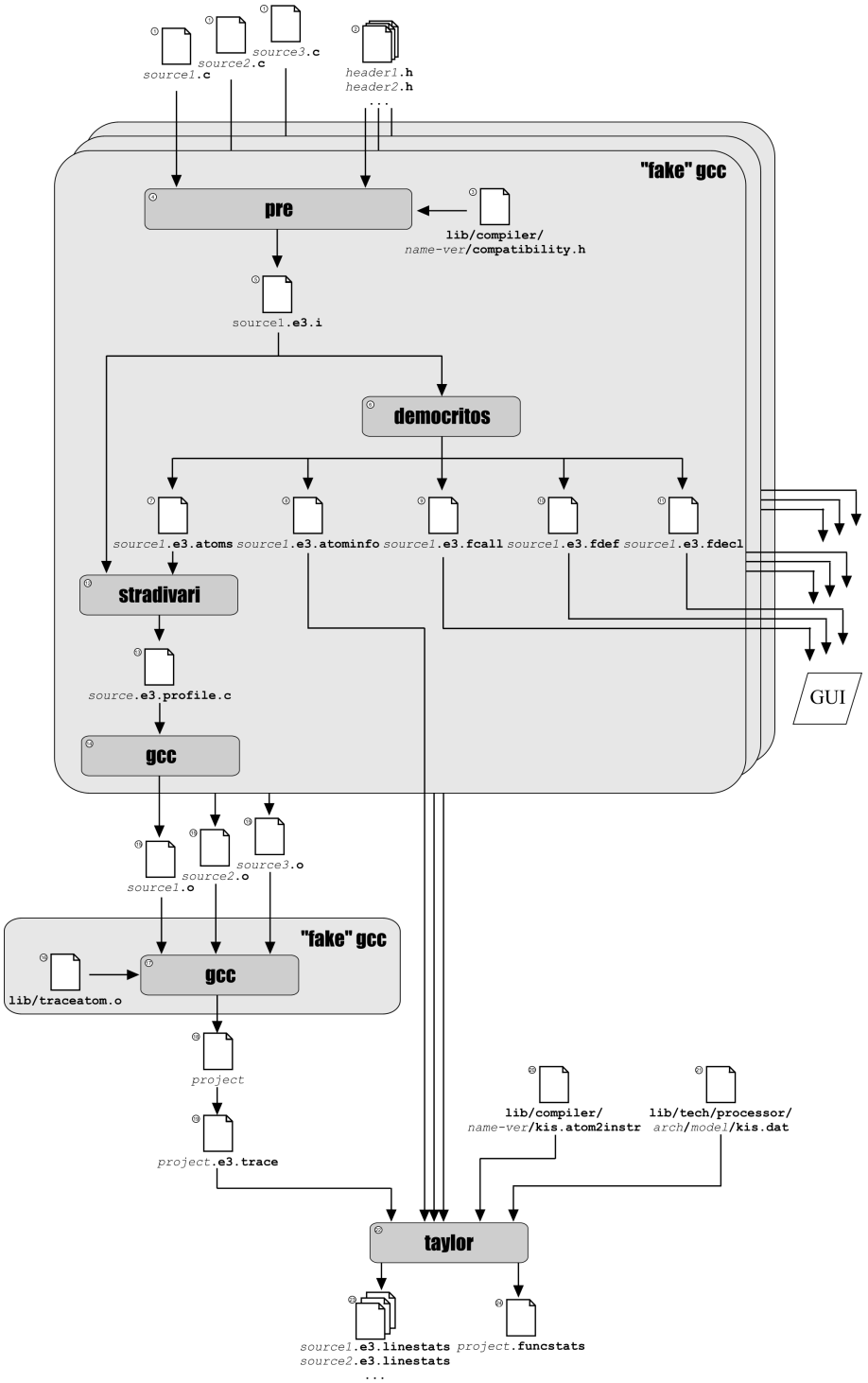

Figure 2: Structure of the current status of our estimation flow.

5 Work plan

5.1 Research phases

1. **literature study**: during this phase we gather all the contributions (regular papers published at conferences and on journals, books, edited books and associated materials) which are related with the research topic. The aim of this phase is to establish which is the state of the art in the research area, which are the main actors involved in the research, and in which conferences the interest is mainly focused on these topics;

2. **estimation flow design**: this phase consists in the design and implementation of the C-language source-level estimation engine. More precisely, this phase consists of the following tasks:

   (a) design and implementation of a C-level project instrumentation and execution tracing flow;

   (b) design of trace data gatherer;

   (c) architectural modelling and salient feature extraction for a few sample processors;

   (d) design and implementation of a tool to collect, extract and query source code information which are relevant for the source code transformation application (e.g., loop depth, loop variables, etc);

3. **optimization flow design**: this phase consists in the design and implementation of the source-level optimizing transformation flow, and involves the following sub-tasks:

   (a) design and implementation of a tool for the automatic determination of critical project code sections, to be optimized first;

   (b) design and implementation of a tool for the automatic determination of the most promising source-code transformations and their respective expected gain ;

   (c) possible automatic transformation application; the amount of automatism that we desire to achieve in the application of source code transformations is still to be determined. We are currently performing a cost/benefit analysis, considering that the novelty of this last approach is moderate, whereas the implementation complexity is harsh;

4. **flow integration**: this phase consists in the full integration of the estimation and optimization flows developed in the previous steps, into the larger-size EU-funded Esprit project named POET. This phase comprises the following tasks:

   (a) design of an interface layer between the source-level estimation flow and the microarchitectural simulator which is already part of POET

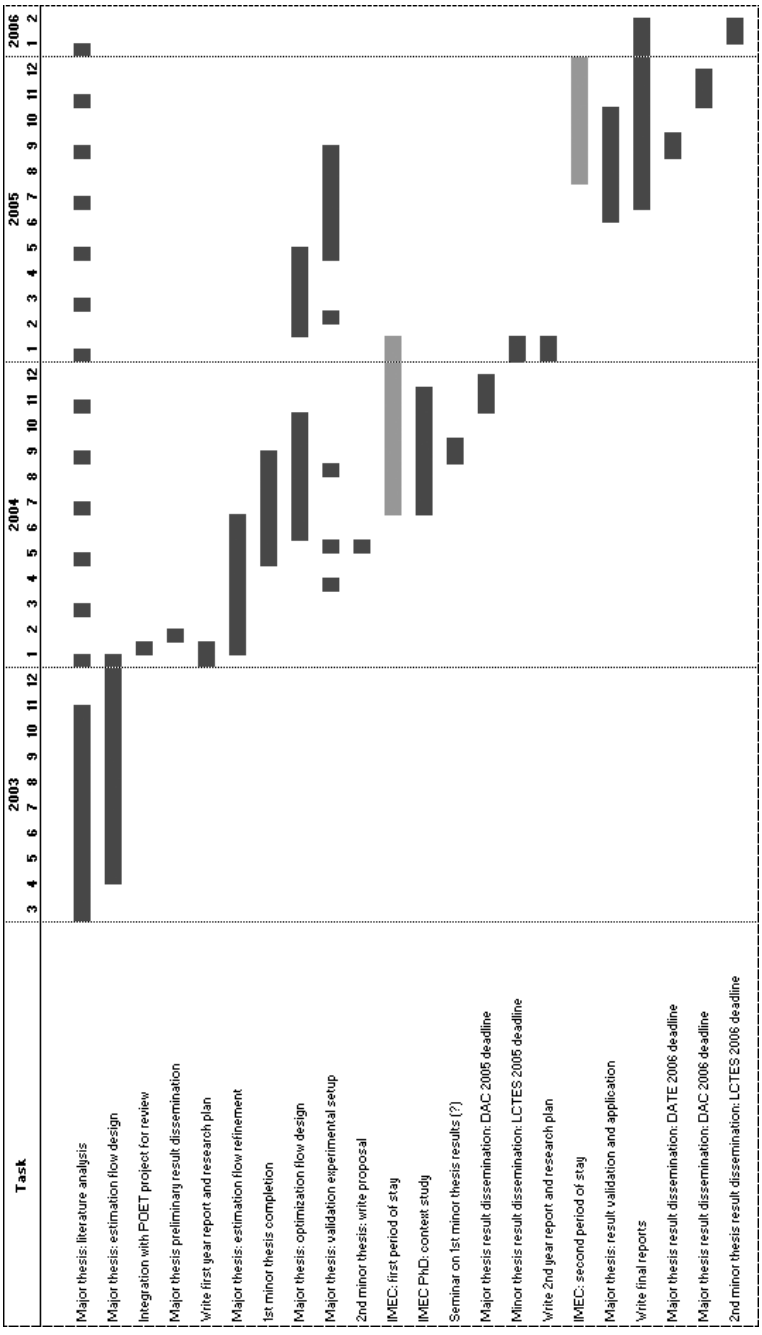

Figure 3: Gantt diagram of the activities involved in this thesis.

(derivation of average consumed energy and execution time from average functional unit absorbed current, average parallelism coefficient, average overhead and nominal execution cycles);

(b) design of the interface layer between the source-level optimization flow and the source-level transformation gain estimation flow (which is already part of POET);

(c) update of the current POET graphic user interface in order to provide a seamless integration of the new tools from the source-level flows into the existing GUI environment.

5. **methodology validation**: the entire methodology will be subject to validation against data obtained from simulation flows (SimplePower, Cacti, Wattch) and, if possible, against data obtained from direct measurement (depends on availability of silicon and dedicated board);

6. **result dissemination**: our dissemination strategy includes the following actions:

   (a) preliminary methodology presentation at conference University booths / Ph.D. Forums;

   (b) intermediate and final result dissemination at conferences, including the Design Automation Conference (DAC) in the USA, the Design Automation and Test in Europe (DATE) Conference, and the ASP-DAC in Asia; and at journals, including IEEE and ACM journals;

   (c) publication of software and documentation on the Internet: We plan to setup a website to provide dissemination of the results obtained during this research. Although all the development technologies we used are open-source, our research consortium agreement does not allow us to disseminate the source code of our products we develop.

5.2 Visits to other research institutions

A visit to IMEC (Interuniversity MicroElectronics Center, Leuven, Belgium) is planned, in two periods of stay, of 7 and 5 months respectively, as indicated on the Gantt chart provided, for a total of one year. Cooperation with IMEC falls in the framework of Imec's *Sandwich Ph.D. system*, under the support of a Marie Curie Fellowship. The title and the abstract of the joint Ph.D. proposal follow.

Evaluation and definition of novel low power methodologies to map jobs/subtasks on low-power heterogeneous processor platforms based on platform simulator

Many embedded multi-media and communication applications exhibit real-time constraints that are either soft (deadline misses should be minimized) or hard but with a relatively low event rate (e.g. video frame rate of 25/30 Hz). The platforms for such applications are becoming more and more flexible including a heterogeneous set of programmable processor cores. Due to the

requirement of low energy consumption that is becoming more crucial even in future technologies (due to increased problems with leakage and low-cost heat removal in packaging) also support for frequency/Voltage scaling is available.

Today, most approaches either rely on static worst-case execution of the jobs and subtasks on such a platform, or on the RTOS scheduler. The static option meets the deadlines but it incurs a potentially large energy penalty because it does not well exploit the available dynamism/data-dependence in the jobs/subtasks. The RTOS option has problems with hard real-time constraints. We propose to look at a novel more hybrid solution where at design-time, several working points are characterized for each of the jobs on the given platform. At run-time, the final scheduling and assignment decision is then made.

The focus of this Ph.D. proposal is to evaluate different methodology aspects based on experiments with real-life multi-media applications with a mix of soft and hard real-time constraints. These experiments are done based on a heterogeneous multi-processor platform simulator. For this purpose also high-level power models will be required for the heterogeneous processor nodes in the platform simulator. Based on this study the main issues that are still not well supported in the existing mapping methodologies will be identified and the highest priority ones will be investigated in terms of systematic method and prototype tool support.

6 Connections with research projects

This doctoral thesis is part of a larger, EU-funded, reseach project, called POET (Power Optimization for Embedded sysTems), more precisely the objectives of this thesis correspond tightly to the ones of work package 2 of the project.

The main objective of project POET is to develop a new design methodology and tool suite for power estimation and optimisation in heterogeneous embedded SoC designs. The key innovation of this approach is to enable design space exploration for low power system architectures, algorithm optimisation and system partitioning - from the earliest design steps seamlessly through to RT level (i.e. to the interface with standard industrial synthesis tools). The POET design framework will operate at each level of abstraction, i.e. algorithmic, hardware/software partition, cycle-accurate RT level. POET tools will manage and optimise all major contributors to power dissipation in large SoC designs such as ASICs, cores and processors, memories, communication and I/O interfaces.

Power management and reduction is most efficiently achieved at the system level, at the stage when algorithms are developed and partitioning is done. Experiments have shown that at one extreme, power reduction of several orders of magnitude can be achieved between the best combination of algorithms and architecture versus non-optimised solutions. POET addresses this opportunity as its main objective the development and integration of methodologies and tools for power estimation and optimisation of combined SW and HW descriptions of SoC systems. POET targets the algorithmic level for both hardware and software as well as the functional RT level of abstraction where power optimisations can offer the largest potential.

The various tools to be developed will operate at different levels of abstraction (including application-level software, algorithmic, cycle-accurate RTL) and will address all major contributors to power dissipation especially in large SoC implementations. Since software and algorithm development of SoCs is mostly done in the C programming language, C will be directly supported as an input language for the system specification. The RT-level optimisation and interface to the back-end design flow will be based on VHDL so that it is consistent and compatible with the established silicon manufacturing design flow.

More specifically about the role of work package 2, that is, this thesis work, the goal is to face the problem of power optimisation of the software components of embedded SoCs. In particular the effort will be focused on source-level analysis and optimisation, by using established compilation environments and/or proprietary compilation environments. Since most of the software for embedded applications is developed using C and since all object-oriented extensions to this language lead to a loss of efficiency and an increase in energy requirements, the methodology will concentrate on the ANSI C language. Extensions and libraries compliant with the POSIX standard will also be considered.

The first part of the project will be devoted to the study of the potential sources of power dissipation (memories, CPU internals, etc.) and their relation

with the basic C operators and constructs. In order to define a strategy to tackle energy optimisation issues, a detailed analysis of the compilation techniques and environments will be considered as a necessary prerequisite. Furthermore, in order to adequately consider technology specific issues, accurate data concerning the power consumption of ARM cores, cache memories and any other additional components are a necessary and mandatory prerequisite. Once the relations among power dissipation sources, source code constructs and compilation techniques have been clarified, the work will address the identification of a set of general optimisation directives as independent from the target microprocessor architecture as possible. Technology specific issues will also be addressed and integrated in the general methodology.

The following step consists in the application of the identified directives on a set of benchmarks in order to validate the effectiveness of each source code transformation. Code transformations will be preliminary applied manually. This phase precedes the definition and the implementation of a tool for the semi-automatic application of the identified source-tosource code optimisations.

The implementation of the cache architecture and operation can have a substantial impact on the overall system power consumption cache size and depth and hit/miss rates, writebuffering type and fetch block size, and cache-flushing decisions all impact the overall system power consumption, silicon area and performance to a considerable degree. In fact, nonoptimal cache design can impact both performance and power consumption to a much greater degree than non-optimised core processor design alone.

6.1 Project partners

6.1.1 Other research institutions

- **Kuratorium OFFIS e.V.** OFFIS, the "Oldenburger Forschungs- und Entwicklungsinstitut fr Informatik-Werkzeuge und Systeme", is a non-profit state-funded research institute associated with the University of Oldenburg in Germany. OFFIS currently employs 100 full time graduate or post-graduate computer scientists. The focus of the current research is on power optimisation of embedded systems, system-level design automation and industry driven low power design training.

- **Politecnico di Torino** The research activity of the EDA (Electronic Design Automation) group in the *Dipartimento di Automatica e Informatica* of Politecnico di Torino is focused on the development of methodologies, algorithms and software tools for the computer-aided design of integrated circuits and systems, with particular emphasis on the design of low power components for usage in portable electronic applications.

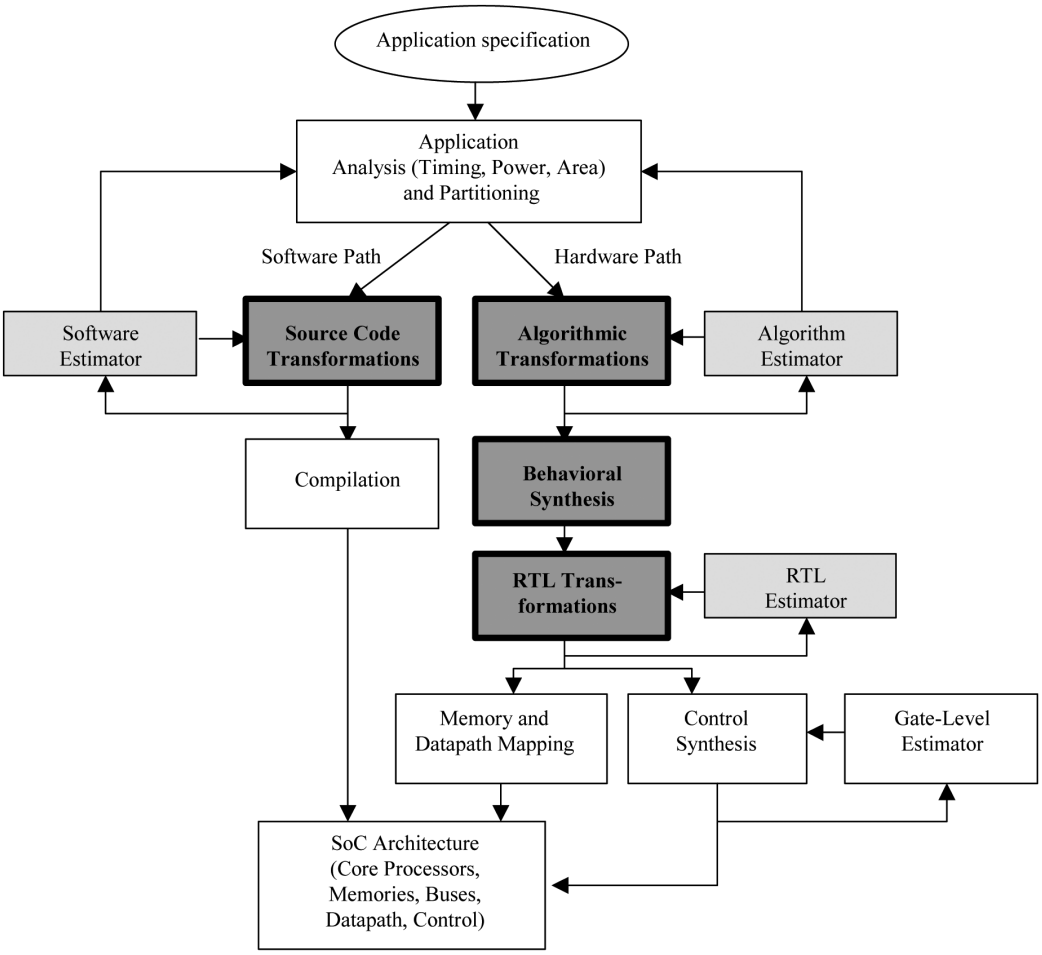

Figure 4: Overall scheme of project POET.

6.1.2 Industrial partners

- **Alcatel SEL AG** Alcatel works in the area of high complexity ASICs. Thus low power is expected to be a crucial issue for all future activities. The developed tools and methods will be used in power critical projects to overcome the lack of commercial solutions. The company foresees that power will be a crucial factor in the design and implementation in new generation ASICs (Systems on silicon), and for this reason it feels that it is of enormous importance to use the estimators and optimisers developed in the POET project. The exploitation of the results during POET Project life-span will be the application of the tools to the design of a complex device. Some simpler modules will be considered as benchmark to pro-

vide feedbacks to the developers. The complexity of the device will be representative of the internal needs of the company. The benefit and advantages offered by the technology will be evaluated thoroughly. After a positive testing and acceptance of the tools, internal qualification will be granted. In this case the exploitation will proceed after the end of the project with the integration of the tools in proprietary design flow. Seminars and training courses regarding power estimation and optimisation will be organized to improve the internal dissemination process. The commercialisation of EDA tools is outside the strategic commitment of the Company, therefore no policy will be applied in this field.

- **ARM Limited** ARM is a provider of 16/32-bit embedded RISC microprocessors. The company licenses RISC processors, peripherals, and system-on-chip designs to electronics companies. ARM provides its partners with solutions comprising cores, tools, platforms, and other Intellectual Property (IP) components required in developing a complete system. The Low power design tools developed in the POET project will enable current and planned ARM architectures to achieve the performance requirements of current and future mobile applications.

- **OSC-OFFIS GmbH** OSC is the marketing channel of OFFIS, and it sells tools for power estimation at the behavioural level. These tools have been packaged together with power estimation of memory usage and power characterisation tools, as well as with a graphical user interface, under the registered trade mark *Orinoco*.

- **BullDAST srl** BullDAST is a a spin-off company of Politecnico di Torino, established in year 2000, with the major purpose of turning into products the prototype EDA tools that have been developed over the years by the EDA group of Politecnico. BullDAST also contributes to the commercial exploitation of the results directly related to the activities covered by Politecnico di Torino.

- **ChipVision AG** ChipVision Design Systems AG, is an EDA SME company, founded in May 2002 as a private share-holder company as spin-out of OFFIS Systems and Consulting GmbH. ChipVision develops and markets system level EDA tools in particular for low power design. It participates to the research and development work, to the training activities, and to the exploitation of results.

- **ATMEL** Atmel's Consumer Imaging division designs complex CMOS imaging ICs. As one of the key performance (low power consumption of the products) and quality (low induced noise due to digital parts activity) contributors of the designs, low power is driving the overall development flow. ATMEL intends to use the power estimation tool and associated methodology to increase the overall performance of the designs developed, through a better and earlier budgeting and exploration

phase of the possible architectural choices. ATMEL's main expected benefit of using specification-level power estimation tools is to get reliable power-related metrics associated with each architectural options of a design, using a streamlined flow.

- **Motorola** Motorola European Wireless Design Center develops baseband chips for mobile devices such as 2G & 3G cellular phones. In order to achieve the low power requirements in ever increasing features of mobile, the SoC design methodology flow from the architecture definition to the chip integration need to be constantly updated and improved, at each level of the development steps, with break through new low power methodologies and tools. The evaluation of tools developed in the POET project will be exploited to reach this goal.

References

[1] V. Tiwari, S. Malik, and A. Wolfe, *Power Analysis of Embedded Software: A First Step Towards Software Power Minimization*, In Proc. of Intl. Conference on Computer-Aided Design, San Jose, CA, USA, 1994;

[2] V. Tiwari, S. Malik, and A. Wolfe, *Compilation techniques for low energy: An overview*, In Proc. of Symp. Low-Power Electronics, 1994;

[3] V. Tiwari, S. Malik, A. Wolfe, M. Lee, *Instruction level power analysis and optimization of software*, Journal of VLSI Signal Processing, pp. 1-18, 1996, Kluwer Academic Publishing, Boston;

[4] J. Russell, M. Jacome, *Software Power Estimation and Optimization for High Performance, 32-bit Embedded Processors*, Proceedings of ICCD '98;

[5] W. Ye, N. Vijaykrishnan, M. T. Kandemir and Mary Jane Irwin, *The Design and Use of SimplePower: a cycle-accurate energy estimation tool*, Design Automation Conference, 2000;

[6] C. Brandolese, W. Fornaciari, F. Salice, D. Sciuto, *An instruction-level functionality-based energy estimation model for 32-bits microprocessors*, Design Automation Conference (DAC-00), Los Angeles, CA, USA, June 2000;

[7] C. Brandolese, W. Fornaciari, F. Salice, D. Sciuto, *Energy estimation for 32-bit microprocessors*, Eighth International Workshop on Hardware/Software Codesign (CODES-2000), May 2000;

[8] C. Brandolese, W. Fornaciari, F. Salice, D. Sciuto, *Fast Software-Level Power Estimation for Design Space Exploration*, The International Conference on Hardware Description Languages (HDLCON-00), San Jose, CA, USA, March 2000;

[9] C. Brandolese, *Retargetable Software Power Estimation Methodology*, Asia Pacific Conference on Hardware Description Languages (WCC-ICDA-2000), Beijing, China, August 2000;

[10] C. Brandolese, W.Fornaciari, L.Pomante, F. Salice, D. Sciuto, *A Multi-Level Strategy for Software Power Estimation*, International Symposium on System Synthesis (ISSS2000), Madrid, Spain, September 2000.

[11] G. Beltrame, C. Brandolese, W. Fornaciari, L. Pomante, F. Salice, D. Sciuto, V. Trianni, *Dynamic Modeling of Inter-Instruction Effects for Execution Time Estimation*, International Symposium on System Synthesis, ISSS2001, Montreal, Canada, October 2001;

[12] G. Beltrame, C. Brandolese, W. Fornaciari, F. Salice, D. Sciuto, V. Trianni, *An Assembly-Level Execution-Time Model for Pipelined Architectures*, IEEE/ACM International Conference on Computer Aided Design, ICCAD2001, San Jose, CA, USA, November 2001;

[13] G. Beltrame, C. Brandolese, W. Fornaciari, F. Salice, D. Sciuto, V. Trianni, *Modeling Assembly Instruction Timing in Superscalar Architectures*, IEEE International Symposium on System Synthesis, ISSS'2002, Kyoto, Japan, October 2002;

[14] C. Brandolese, W. Fornaciari, F. Salice e D. Sciuto, *Library Functions Timing Characterization for Source–Level Analysis*, IEEE/ACM Conference on Design Automation and Testing in Europe, DATE'2003, Munich, Germany, March 2003;

[15] C. Brandolese, W. Fornaciari, L.Pomante, F. Salice, D. Sciuto, *Source-Level Execution Time Estimation of C Programs*, IEEE International Workshop on Hardware Software Co-Design, CODES2001, Copenhagen, Denmark, April 2001;

[16] E. Y. Chung, L. Benini, G. De Micheli, *Source Code Transformation Based on Software Cost Analysys*, ISSS 2001, Montreal, Canada, October 2001;

[17] C. Brandolese, W. Fornaciari, F. Salice, D. Sciuto, *The Impact of Source Code Transformations on Software Power and Energy Consumption*, IEEE Journal of Circuits, Systems and Computers, Vol.11, p. 477-502, 2002;

[18] D. P. Scarpazza, *POET – Power Optimization for Embedded Systems – Software Path*, Technical Report – February 2004, Politecnico di Milano - Technical Report: 2004.11, 2004;