

An Energy Conscious Methodology for Early Design Exploration of Heterogeneous DSPs

Marlene Wan, Yuji Ichikawa*, David Lidsky, Jan Rabaey

EECS Department, University of California at Berkeley
Berkeley, CA. USA

* Sharp Corporation, Japan

Abstract

In this paper, we introduce an energy-conscious methodology to guide algorithm partitioning and mapping of embedded DSP applications onto heterogeneous architecture components. The methodology supports both realistic algorithm-architecture (simultaneous optimization at different abstraction levels) as well as hardware-software co-design (optimization over various architectural alternatives). Macro-model based predictors are used to provide early feedback on the impact of design selections and partitions. A case study is presented to demonstrate the methodology flow.

1. Introduction

With the exponential improvement in integrated circuit density, systems-on-a-chip are rapidly becoming a reality. Circuits, combining a wide variety of macromodules including core processors, DSPs, programmable logic, embedded memory, and custom modules have been reported by a number of companies [4]. The majority of these designs target the embedded computing market with distinguishing features such as stringent constraints on the power, delay (or performance), and area (PDA) metrics as well as an equal balancing between software and hardware implementation.

The high-level of integration and the myriad of architectural options it enables create a need for an architectural exploration environment that provides the designers with meaningful design guidance and a means to evaluate different choices and their impact in the early phases of a project. One of the prime requirements of such an environment is that it can effortlessly deal with the multiple levels of programming granularity [17] that are becoming prominent in today's embedded systems (ranging from general-purpose embedded microprocessors to programmable logic modules). In addition, in order to make use of the opportunities offered by these architectural options, a reformulation of the original algorithm is often needed. The design methodology that optimizes algorithm and architecture in concert is termed *algorithm-architecture co-design* in the rest of the paper.

In this paper, we propose an interactive design exploration environment for heterogeneous embedded systems that supports both realistic algorithm-architecture (simultaneous optimization at different abstraction levels) as well as hardware-software co-design (optimization over various architectural alternatives). Macro-model based predictors are used to provide early feedback on the impact of design selections

and partitions.

Of the limited design environments which address the issues raised above [6], the majority of them focus almost entirely on the timing and area, with only casual references to energy or power [5]. As power dissipation being one of the major constraints in today's embedded systems, a meaningful exploration environment should consider energy, delay, and area on the same footing. The introduction of an energy-conscious system-design methodology is another contribution of this paper. Since analysis of both area and performance cost functions is rather well understood, the bulk of our attention in this paper will be devoted to the power models and predictors.

While the proposed methodology is general in nature and valid for a wide variety of embedded systems, we have chosen to apply it to one particular architectural model that was the original motivator for the development of this software methodology. The architecture of interest [1] combines a general-purpose processor with a reconfigurable network of accelerators, which are programmable at different levels of granularity (reconfigurable logic, reconfigurable data paths, weakly programmable arithmetic units). The intended application domain of these heterogeneous computing systems is real-time embedded DSP, such as encountered in wireless, multimedia, and consumer applications. A short description of this architectural template is presented in Section 2 of the paper to provide some context and motivation.

An interactive exploration methodology for the target architecture is proposed in section 3, and the section also highlights the various components of the overall design flow. An extensive case study — a VSELP voice coder for wireless applications — demonstrates the effectiveness of the approach in Section 4. The paper concludes with the current status and future challenges.

2. An Architectural Model for Heterogeneous Computing Systems

The majority of the embedded applications contain a substantial amount of computational-intensive DSP algorithms. This translates into a quest for architectures that provide high performance at low energy. In addition, time-to-market, cost, or application requirements often dictate that these implementations must be highly programmable.

An aggressive DSP architecture that strives to combine high performance, low power, and high degree of program-

mability, is shown in Figure 1. This architecture, which is the focus of the Berkeley Pleiades project [14], consists of a microprocessor and a collection of hardware accelerators (programmable at different levels of granularity) connected through a reconfigurable network.

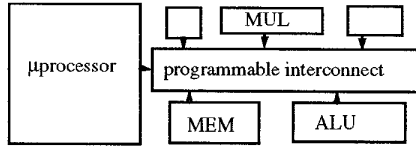


Figure 1 Low-power reconfigurable DSPs

The architecture combines two very distinct models-of-computation: control-driven computation on the general-purpose processor and data-driven computing on the clusters of co-processors. The goal of the architectural exploration process is to partition the application over these two paradigms so that performance and energy dissipation constraints are met (during the compilation process), or to determine the type and number of co-processors needed to adequately support a set of applications (during the architectural design process). The exploration environment can also be used to optimize the application description to better match the properties of the underlying architecture. The target applications are real-time DSP algorithms specified in a subset of the C/C++ language. The usage of high-level languages is an essential requirement for an effective DSP fast-prototyping environment [13].

3. Proposed Methodology

The flow of the exploration methodology is presented in Fig. 2 and outlined below. After the introduction of some terminology and a short overview of the overall flow, a detailed description of each of the components is given.

3.1 Terminology

Computational kernel:

The computationally-intensive part of an algorithm.

Most often, kernels represent the inner-bodies of frequently executed loops. In most DSP algorithms, the majority of the computational complexity is centered in just a few kernels.

Macromodel:

A quantified view of the cost functions of the design module as a function of its parameters and constraints.

A macromodel can take on many forms: it can be represented by a fixed number, a table, a set of equations, an invocation of a tool such as an estimator, or it can be composed of a set of other macromodels. A macromodel presents a uniform entry of inputs, and return of outputs, regardless of the method used to calculate the outputs. It is up to the environment to interpret the input parameters, find the proper macro-

model, evaluate the macromodel, and provide the estimate(s) to the user in a meaningful manner.

The main benefit of macromodels is that they may be used as a black box so that a designer interacts with designs in the same manner regardless of the subcomponents. For example, using macromodels, the effects of changing a hardware library can be instantly evaluated without rewriting the behavioral or structural description.

3.2 Basic Design Flow

In this section, the numbers in parentheses refer to the ones in Fig. 2.

The initiation of the design process requires the establishment of a first-order baseline model of the algorithm complexity and bottlenecks. Such a model allows for the selection and execution of architecture-independent optimizations (1). As architectural choices yet have to be made, this model assumes the presence of a “virtual architecture” with some generic operator costs attached to it and minimal constraints. Optimizations at this stage only address either win-only situations or order-of-magnitude improvements, so that absolute accuracy is not that important.

This optimization and refinement process is repeated until a satisfactory formulation of the algorithm is obtained. This allows for the architectural mapping and partitioning

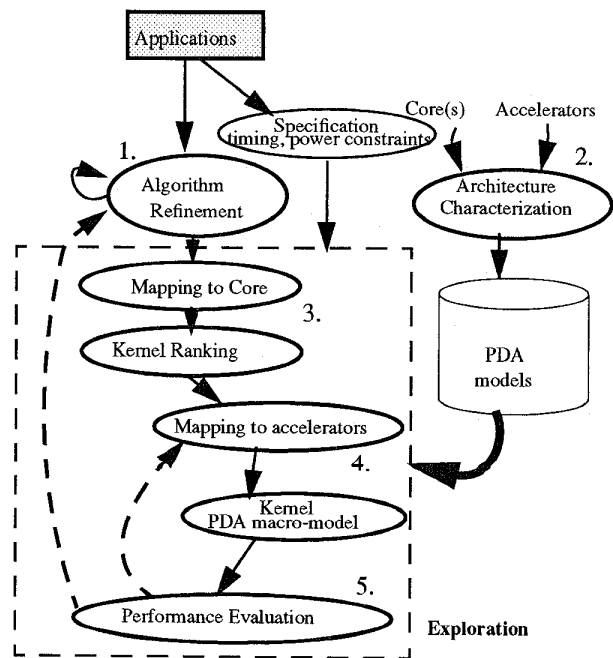


Figure 2 Basic flow of proposed exploration methodology.

process to be entered.

To be meaningful, the partitioning process should be based on realistic bottom-up information regarding the cost of implementing functions and operations on the different architectural choices. Our design-exploration methodology relies extensively on the availability of PDA (Power-Delay-Area) macromodels for all components in its architectural library (2). The methods employed in each of these models vary depending upon the type of the module and the desired accuracy. While the absolute accuracy of these characterizations is not crucial, it is important that bounds on the prediction accuracy are known. "Improvements" that fall within the noise level of the estimations should be treated wearily.

The architecture partitioning and mapping process is started by establishing an initial solution. Given the attractiveness of a pure software implementation, we have adopted a "software-centric" approach that assumes that the whole algorithm is initially mapped onto the core processor (3). This establishes how close such a solution adheres to the design specifications and helps to establish the design bottlenecks. A rank ordering of the dominant compute kernels is established. Dominant kernels are evaluated in order of importance. If a hardware implementation is deemed worthwhile, a repartitioning of the design is established (4).

To facilitate the interactive exploration process, a simple and effective visualization of the trade-off's is essential (5). The hyper-spreadsheet approach, introduced in PowerPlay macromodeling environment [10], accomplishes exactly that goal.

The macromodeling approach advocated in this paper allows for the simultaneous exploration of a number of different cost functions. While multi-dimensional search is an essential requirement for every exploration environment (changing one cost function is likely to have (un)desirable effects on another), we will direct our focus on the power dimension in the rest of the paper.

3.3 Components

The details of each components in the methodology is given in the following paragraphs.

Algorithm Characterization and Refinement

Modern DSP algorithms such as voice and video CODECs combine regular but computation-intensive kernels (typically the dominant factor) with irregular control functions. While the latter are best implemented on a traditional processor architecture, the former present ample opportunity for design optimization by exploiting hardware accelerators.

It is important to identify potentially power-hungry portions of an algorithm as early as possible. The *inherent computational complexity* (counts of basic operations and memory accesses) is a meaningful measure to identify dominant kernels [11]. This information can be obtained through a combination of dynamic profiling (loops and conditionals) and static analysis (basic blocks) of the high-level specifica-

tion. Since operations don't have a uniform cost, a weighted sum (where the weights indicate the energy allocated to each operator in the abstract implementation model) is calculated and aggregated at either the function or basic block level to indicate the power consumption trend within the application.

Based on this information, the designer can rewrite the code to either reduce the algorithm inherent cost, or extract kernels into procedure calls. These procedure calls are then considered for potential hardware acceleration.

The algorithmic refinement process is based on a combination of commonly available tools. The algorithm is simulated with appropriate input vectors (representing standard operation), and profiling information is gathered at the basic block level (e.g. using the gcc -a -g options combined with post-processing). The profiling frequency is back-annotated to the source code to provide a first-order complexity-breakdown. Combining the basic-block execution frequencies of the profiler with a breakdown of the blocks into basic operations, as provided by standard compiler parser front-end's [7], generates a detailed and illustrative overview of the distribution of the algorithm complexity over basic operators and memory accesses.

Architecture Characterization

A successful system modeling technique relies not only on the use of macromodeling, but also on a careful characterization of the underlying architectural choices. We introduce several modeling methods for the programmable devices in the target architecture and discuss how macromodels can be built using these primitives.

Instruction-level modeling has emerged as the preferred way for characterizing the energy consumption of general-purpose cores [19]. The energy base cost of each instruction is obtained by either physical measurement (e.g. [19]), physical or RTL-level simulation, or vendor supplied technology notes [18]. Each instruction base-cost includes a scaling factor introducing the effects of frequency and voltage. Given a piece of assembly code, the total energy consumed is the sum of the base costs of the executed instructions. More accuracy can be obtained by adding inter-instruction effects, as well as corrector factors for pipeline stalls and cache misses.

Currently, we have evaluated the instruction base-costs and pipeline stall effects for several ARM cores [8] through physical measurements performed on ARM evaluation boards. In addition, a profiler (included in ARM SDK2.1) is modified to record the energy at each instruction boundary to trace the dynamic behavior of the algorithm.

The power-characterization of the accelerator modules is somewhat more complex due to the variability of the units, the different levels of programmability and influence of hard to determine factors such as signal correlations. On the positive side, moving components from soft- to hardware are only worthwhile when resulting in substantial improvements

— some inaccuracy is hence acceptable. Furthermore, adequate macro-modeling techniques have been developed for parameterizable functional modules such as memories, multipliers, ALUs, etc. [9]. Analytical models of the effective switching capacitance C_{eff} are derived from circuit-level simulations, hence including effects such as short-circuit currents. Given the unknown nature of the signal statistics in a given accelerator configuration, a (pessimistic) white-noise signal distribution is assumed during characterization. The total energy spent by a given functional module is then modeled as:

$$Energy_{FU} = C_{eff} \times N \times V_{dd}^2 \quad (1)$$

where N equals the number of accesses of the module. For fine-grained programmable units such as embedded FPGAs, it is essential to develop characterized libraries of macro-modules, lest time-consuming mapping and analysis steps have to be performed during exploration.

The reconfigurable interconnect module can consume considerable energy. Models for estimating interconnect power have been proposed in [12]. Given the size and number of the functional modules in the accelerator network, it is possible to predict the average length and capacitance of each wire. The total energy can then be estimated using the following expression:

$$Energy_{INTER} = \left(\sum_{i \in network} C_i \times V_{dd}^2 \right) \times N_{inter} \quad (2)$$

with N_{inter} the number of accesses to the network.

Parameterizable macromodels for various hardware kernels (such as dot-vector products, FFTs, DCT/IDCTs, etc.) can also be developed as a function of the parameter set P (which can be used to derive model parameters such as N and N_{inter}) as well as the energy predictors for functional units and interconnect.

Base Cost and Kernel Ranking

Given the desirability of a software implementation, it is a natural choice to first explore the power and timing performance of the application when implemented entirely on the processor core. This can be obtained by combining the refined algorithm obtained from step 1 and the microprocessor energy (timing) estimation tool in step 2. These costs establish a baseline to measure the impact of architectural optimizations. It can also be used to evaluate compliance with timing and energy constraints and the range of improvement that has to be obtained.

Because of the inefficiency of most general purpose cores, some of these constraints may not be satisfied. To identify areas of substantial improvement, a relative rank ordering of the dominant kernels is established. Using the dynamic instruction-level code profiler, a percentage breakdown of the energy consumed by each function (and basic block) is extracted and presented in a hierarchical call-tree format.

The resulting ordering can be used to guide the ensuing optimization strategy (in this context, local optimizations can be considered to be global as well for both speed and energy, i.e. optimization of a single kernel directly translates into a global optimum). Furthermore, it establishes a ground rule for the amount of possible improvement that can be made in the style of Amdahl's law.

While this quantitative ranking is surely very instructive, some other factors have to be taken into account when determining what kernels are prime candidates for acceleration. Some other algorithmic properties determine if a kernel should be disqualified or given more importance. For example, our architectural template supports only "data-flow"-like computation. Therefore, any computation with distinctive control-flow (if-then-else) properties cannot be efficiently implemented in hardware. Similarly, computations with very irregular or data-dependent memory accesses do not translate in effective hardware implementations either. We call these *hardware-repellent kernels* and they are disqualified from the candidate list.

At the end of this step, we have a list of candidate kernels for hardware mapping and their ranking in terms of potential power saving.

Architectural Partitioning and Kernel Mapping

Using an iterative refinement process, the most costly kernels are mapped to the hardware accelerators. The power performance of the kernels are re-evaluated by building a macromodel of the kernel and using information obtained from the architecture characterizations. For estimation purposes, a number of mapping strategies can be pursued:

- "Relaxed" or fast mapping delivers well-established bounds on speed and energy for a given kernel. These techniques were developed and used effectively in the high-level synthesis community [15].
- Simplified mapping taking only a couple of algorithmic properties into account (such as operation and memory access counts).
- Libraries of well-known kernels.
- Manual mapping.

Our current approach uses a mix of the above. The use of the exploration tool described in the paragraph below actually makes a manual exploration approach a plausible and even attractive choice.

Performance Evaluation and Exploration Environment

The goal of the exploration environment is to measure compliance with the overall design goals and to determine the next move in the architectural partitioning and selection process. In its current incarnation, this process is highly iterative and interactive. Proper feedback and ease of introduction and evaluation of architectural choices is a prime requirement if such an environment is to be successful. Design experiments have shown that a spreadsheet-like tool is the preferred environment for exploration and trade-off analysis, which explains our selection of the PowerPlay

hyper-spreadsheet [9] as the prime workbench for our architectural exploration. Especially, PowerPlay’s use of macro-modeling to evaluate cost functions is considered to be a key enabler for the trade-off analysis required by this methodology.

The main feature of PowerPlay are its hyperlinked spreadsheets. Each row in a spreadsheet represents an instantiation of a macromodel. Evaluation of these models proceeds along very distinct roads depending upon the type of the model: evaluation of equation, table look-up, execution of remote analysis tools (including standards such as Mathematica or Matlab), and through the execution of special-purpose estimator or predictor programs. Hierarchy is supported through the use of the hyperlinks: a macromodel for a composite module is simply represented by a spreadsheet, instantiating the composing elements.

The interactive exploration now proceeds along the following track. A conceptual representation of the algorithm under study is given by the call tree of the algorithm with all possible kernels as leaf nodes (Figure 3). At the start, each node is represented by a macromodel, corresponding to a pure software implementation. During the refinement process, the kernels will be one at a time replaced by a model representing a hardware implementation. The new cost is propagated up to the top level and the overall saving can be re-evaluated. Many more kernels can be mapped and their effect on the overall performance can be evaluated and judged. Once the overall timing constraints are met, other architecture parameters can be tuned to optimize the energy consumption (e.g. slow down the clock). If the constraints are not met after the exploration, another algorithmic optimization step is necessary after which the partitioning flow needs to be re-iterated.

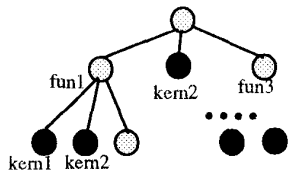


Figure 3 Conceptual representation of the design

To facilitate the trade-off analysis and to make the process of architectural mapping more amenable, two new concepts have been introduced in PowerPlay, the *generic* and the *domain*. A generic entry represents a placeholder for a macromodel. For example, the generic “adder” can be seen as standing for an alias pointing to a more specific implementation. A domain associates generics with their corresponding macromodel. For instance, in a “high-speed” domain, the alias “adder” will refer to a “look-ahead adder”. In our exploration, pointers to kernels are defined as generics. Choosing a particular domain (software, hardware, FPGA) links these aliases to different implementation styles and hence models.

4. An Exploration Case Study

This section demonstrates how the proposed methodology is used to partition and map a real-time DSP algorithm to a heterogeneous architecture template.

This case study demonstrates snapshots of the exploration methodology, as used in mapping the 16-bit fixed point VSELP encoder to a low-power implementation. The specification requires the encoder to process 50 speech frames per second.

Algorithm Characterization and Refinement

The algorithm written in C is simulated and the basic-block execution-frequency is gathered. To get a first-order approximation of the inherent computational complexity of the algorithm, basic operators are extracted (addition, multiplication, comparison, array access) using the parser front end and combined with the execution frequency. Each operator is associated with weights which reflect potential power consumption (obtained from the UCB low power library). Fig. 4 shows the function-level breakdown for the algorithm and the back-annotated source file.

It can be observed in Fig 4 that the function *QuantizeGain* represents a considerable complexity (8.02% of the application) which may imply the presence of potential kernels. The annotated source file indicates the existence of such a loop (86.46% of the function). The loop can then be decomposed into several simple (and regular) base kernels such as “*vector_sum_with_scalar_multiply*”.

Architecture Characterization

The target architecture is composed of an ARM-compatible core combined with dedicated accelerators [2]. Instruction-level energy costs for the core processor were obtained through physical measurements. The Pleiades co-processor modules can be reconfigured to perform different kernels by configuring the interconnects and setting the module parameters (Fig. 5). The co-processor and interconnect modules were characterized using the EPIC PowerMill and TimeMill

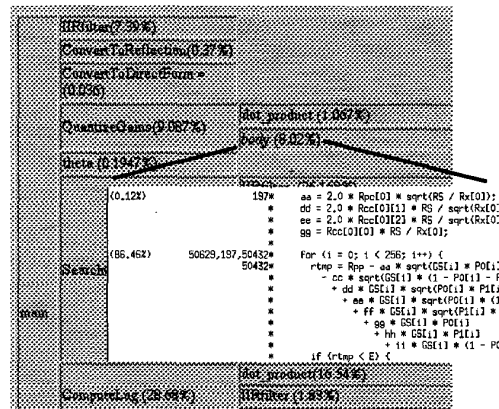


Figure 4 Complexity breakdown for algorithm refinement

Number and Type of Satellite Processors:
3 Address Generators, 3 Memories, 1 MAC/MUL and 1 ALU

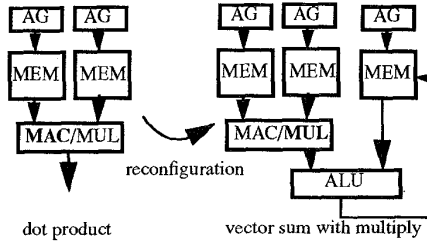


Figure 5 Example of Pleiades accelerators and reconfiguration

tools. For this case study, it is assumed that the interconnect between each satellite processors that all interconnections have equal delay and power.

Base Cost and Kernel Ranking

After the algorithm refinement, the algorithm is compiled to the specified microprocessor and simulated using the ARM instruction-level energy profiler. For the baseline cost, the core is set to run at 1.5V, 169MHz. The total energy, timing, and the energy percentage breakdown of each function is listed in Fig. 6. The hierarchical call graph is imported into PowerPlay as a hierarchy of macromodels and generics. Each node is given an initial cost, which is a linear function of the parameter SPEED since the energy of the software is proportional to the core speed.

Based on the energy percentage breakdown in our example, the top 4 energy consuming kernels are:

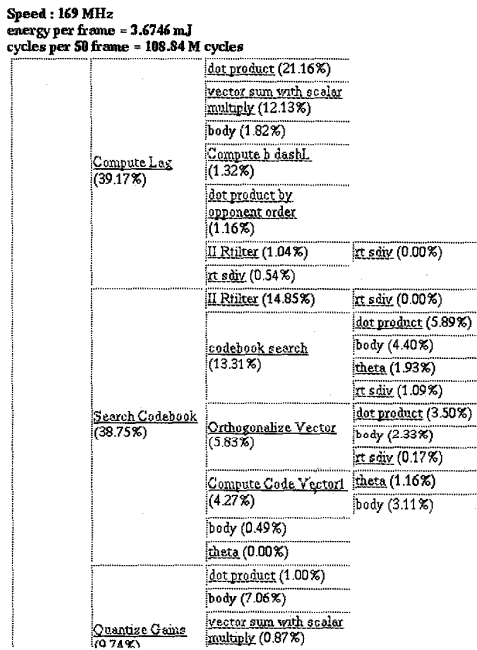


Figure 6 Energy breakdown of VSELP (implemented in software)

dot_product in ComputeLag: 21.16%
llRfilter in SearchCodebook: 14.85%
vector_sum_with_scalarmul in ComputeLag: 12.13%
function_body in QuantizeGain: 7.06%

However, not all of the kernels listed above are suitable for hardware execution, and are hardware-repellent. The front-end informs that control-flow like computations (i.e. if-then-else) are called more than 1024 times per frame on average in one of the highest-ranked kernels (*function_body* of *QuantizeGain*). No control flow computations are present in the other three kernels. The hardware-repellent kernel is either eliminated from the kernel list or becomes a prime candidate for algorithmic transformation (for instance, by replacing the if-then-else by mux operations). The next step is to map the next top kernel - *dot_product* in *ComputeLag*.

Kernel Mapping and Partitioning

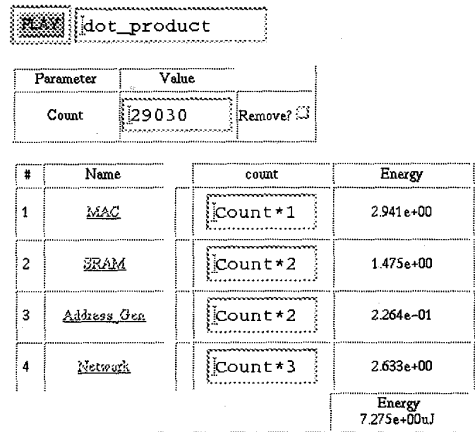


Figure 7 A kernel macromodel viewed in PowerPlay

The inner loop of *dot_product* can be implemented in hardware using two memory accesses, two address generations, one multiple-accumulate, and the associated interconnection. The hardware macromodel for the kernel is shown in Fig. 7. The parameter *Count* represents the number of iterations of the inner-loop.

According to the dynamic analysis, the inner-loop of *dot_product* is executed 29030 times in *ComputeLag*, which serves as the input parameter to the *dot_product* macromodel.

Performance Evaluation and Exploration

A new domain, Pleiades, is created to associate a DOT-PRODUCT generic with the above implementation. Figure 8 shows the domain switch from software to Pleiades.

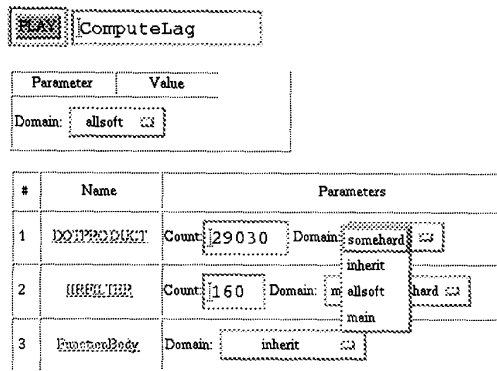


Figure 8 Changing domains in PowerPlay

The new energy and timing for the application is 2.904 mJ/frame and 87.70 cycles/sec respectively. Based on the new energy cost and kernel ranking, suitable kernels can be re-mapped and re-evaluated and the impact on the overall performance can be determined. This process can be repeated till a suitable design is obtained.

In our case study, the following kernels in all functions are moved to the accelerators:

IIRfilter, *dot_product*, *vectorsum_with_scalarmul*, *Compute_CodeVector*, *Compute_bL*, and *theta*.

After the repartitioning, the core processor has to run only 25.64 M cycles per second. This allows us to slow down the processor from 169 MHz to 33 MHz. The energy is scaled accordingly (by changing parameter SPEED). Fig. 9 shows the more than an order of magnitude reduction in energy after repartitioning and reconfiguring of the core processor speed.

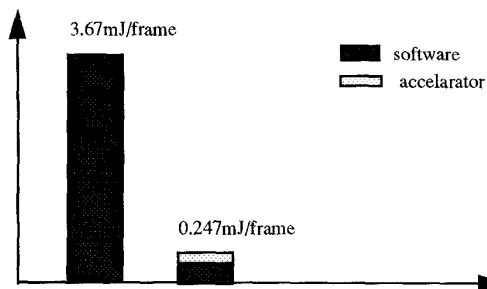


Figure 9 Energy before and after the repartitioning

5. Conclusion and Future Challenges

We have demonstrated an energy-conscious methodology for early design exploration which emphasizes high-level bottleneck detection, and advocates an early prediction of the impact of different mapping choices using macromodelling.

Future work will address to the following challenges:

(1) apply exploration results to automatic estimation and optimizations of the system software and hardware.

(2) use more sophisticated algorithm properties to investigate the suitability of mapping a kernel to a specific architecture.

(3) import more architecture models into the exploration tool.

6. Acknowledgments

We would like to acknowledge DARPA's support for the Pleiades project (DABT-63-96-C-0026). The authors would like to thank the Pleiades team for coming up with great ideas and putting in hard work for the test chip. We also would like to thank Edmond Lau for helping to evaluate the ARM processors.

7. References

- [1] A. Abnous and J. Rabaey, "Ultra-Low-Power Domain-Specific Multimedia Processors," *Proc. of the IEEE VLSI Signal Processing Workshop*, San Francisco, October 1996.
- [2] A. Abnous et al. "Evaluation of a Low-Power Reconfigurable DSP Architecture", submitted to the *5th Reconfigurable Architecture Workshop*.
- [3] Advanced RISC Machines, ARMulator version 2.10.
- [4] J. Borel, "Technologies for multimedia systems on a chip", *1997 IEEE International Solid-State Circuits Conference*, pages. 18-21.
- [5] B.P. Dave, G. Lakshminarayana and N. K. Jha, "CONSYN: Hardware-Software Consynthesis of Embedded Systems", *Design Automation Conference*, pp.703-708, 1997.
- [6] G.De Micheli and R. K.Gupta. "Hardware/Software Co-Design", pages 349-365, *Proc. of the IEEE*, Vol 85, No.3, March 1997.
- [7] Edison Design Group, "C++ Compiler Front End".
- [8] D. Jagggar, editor. *Advanced RISC Machines Architectural Reference Manual*. Prentice Hall, 1996.
- [9] P. Landman and J. Rabaey, "Black Box Capacitance Models for Architectural Power Analysis", *Proc. of the International Workshop on Low Power Design*, pp. 165-170, April 1994.
- [10] D. Lidsky and J. Rabaey, "Early Power Exploration -- a World Wide Web Application", *Proc. Design Automation Conference*, Las Vegas, NV, June 1996.
- [11] R. Mehra, et al. "Algorithm and Architectural Level Methodologies for Low Power", pages 335-362, in Rabaey (14).
- [12] R. Mehra, "High-Level Estimation and Synthesis Techniques for Low-Pwer Design", Doctorial Thesis, May, 1997.
- [13] P. G. Paulin et al. "Embedded Software in Real-Time Signal Processing Systems: Application and Architecture Trends", pages 419-435. *Proc. of the IEEE*, Vol 85, No. 3. March 1997.
- [14] The Pleiades project homepage (<http://infopad.EECS.Berkeley.EDU/research/reconfigurable/>)
- [15] M. Potkonjak and J. Rabaey, "Exploring the Algorithmic Design Space using High Level Synthesis", *Proc. of IEEE Workshop on VLSI Signal Processing VI*, pp.123-131, 1993.
- [16] J. M. Rabaey and M. Pedram, editor. *Low Power Design Methodologies*. Kluwer Academic Publishers, Massachusetts, 1996.
- [17] J. M. Rabaey, "Reconfigurable Computing: the Solution to Low Power Programmabl DSP", *Proc. to 1997 ICASSP Conference*, Munich, April 1997.
- [18] Texas Instruments Application Reports, <http://www.ti.com/sc/docs/psheets/appnote.htm>.
- [19] V. Tiwari, S. Malik, A. Wolfe, and T.C. Lee, "Instruction Level Power Analysis and Optimization of Software", *Journal of VLSI Signal Processing*, 1996.

NOTES
