Dynamic Data Type Refinement Methodology for Systematic Performance–Energy Design Exploration of Network Applications

Alexandros Bartzas†, Stylianos Mamagkakis†, Georgios Pouiklis†, David Atienza⋆,
Francky Catthoor‡, Dimitrios Soudris†, Antonios Thanailakis†
† VLSI Design and Testing Center–Democritus Univ. Thrace, 67100 Xanthi, Greece.
Email:{ampartza,smamagka,gpouikli,dsoudris,thanail}@ee.duth.gr
⋆DACYA/UCM, 28040 Madrid, Spain & LSI/EPFL 1015 Lausanne, Switzerland.
Email: datienza@dacya.ucm.es
‡IMEC vzw, Kapeldreef 75, 3001 Heverlee, Belgium. Email: Francky.Catthoor@imec.be
Also professor at the Katholieke Univ. Leuven, Belgium

Abstract

Network applications are becoming increasingly popular in the embedded systems domain requiring high performance, which leads to high energy consumption. In networks is observed that due to their inherent dynamic nature the dynamic memory subsystem is a main contributor to the overall energy consumption and performance. This paper presents a new systematic methodology, generating performance-energy trade-offs by implementing Dynamic Data Types (DDTs), targeting network applications. The proposed methodology consists of: (i) the application-level DDT exploration, (ii) the network-level DDT exploration and (iii) the Pareto-level DDT exploration. The methodology, supported by an automated tool, offers the designer a set of optimal dynamic data type design solutions. The effectiveness of the proposed methodology is tested on four representative real-life case studies. By applying the second step, it is proved that energy savings up to 80% *and performance improvement up to* 22% *(compared to the original implementations of the benchmarks) can be achieved. Additional energy and performance gains can be achieved and a wide range of possible trade-offs among our Pareto-optimal design choices are obtained, by applying the third step. We achieved up to* 93% *reduction in energy consumption and up to* 48% *increase in performance.*

1. Introduction

In the last years, there is a trend toward networks and network applications implemented with the use of embedded consumer devices. The complexity of modern wired and wireless networks combined with the increased interaction with the environment (e.g. in wireless networks) has increased the dynamism of the data access pattern.

The dynamism of network applications depends on various factors. One of them is the network traffic. Depending on that, the behavior of functions, as well as the number of times they are executed, differ. The amount of stored data, needed for these functions, varies as well. Thus, dynamically adjustable storage size is a necessity, allowing for freeing of memory when it is no longer needed. Moreover, a static memory allocation at compile time is not an option in the case of most modern network applications, because they do not have a static data access behavior, but rather a dynamic one. This fact means that a static memory allocation at compile time is not efficient at all, because the worst case situation has to be assumed in the beginning and implemented for the whole execution time. Therefore, dynamic memory allocation and management is required (especially in embedded systems). This leads to an increased reliance on Dynamic Data Types (DDTs from now on), the structures, which allow data to be dynamically allocated and deallocated at run-time and provide an easy way for the designer to connect, access and process data [17]. DDTs (with the most common examples being the single and doubly linked lists) can efficiently cope with the variations of run-time needs (e.g. network traffic, user interaction) and the massive amounts of transferred and stored data.

Inefficient use of DDTs, results in performance losses by adding computational overhead for the internal DDT usage mechanisms. Energy consumption is the limiting factor in the amount of functionality that can be placed in embedded systems. DDTs consume energy by accessing data in the memories, where they are stored. These two factors cannot be optimized with the same DDT implementation (i.e. a fast DDT is not always the most energy efficient) and the designer must be able to choose a balanced DDT implementation, achieving in every case the required performance,

3-9810801-0-6/DATE06 © 2006 EDAA

while minimizing energy consumption. Nevertheless, great memory footprint size gains in comparison to a statically allocated compile-time memory solution, can be achieved.

The final decision, about the optimal combination of the different DDTs implemented in the network application, is influenced by the complex dynamic behavior. Therefore, no general, domain-specific optimal solution exists but only custom, application-specific ones. On top of that, the different configurations available to networking applications add one more layer of complexity and demand further DDTs customization to achieve optimal results. Thus, the decision should be in accordance to both the application's dynamic behavior [3] and the network configuration. A systematic, step-by-step methodology is needed to help the designer to make the right trade-off choice for each DDT in the networking application.

In this paper, we present a systematic methodology, supported by atool, to perform DDT refinement of any given network application, with any network configuration, using Pareto-optimal execution time-energy consumption trade-offs. The methodology offers a plethora of optimal solutions based on Pareto curves, representing an optimal implementation only if no other implementation has better results in all metrics explored (a point is said to be Pareto-optimal, if it is not longer possible to improve upon one cost factor without worsening any other [4]). Additionally, it offers significant gains in total simulation time by reducing the exploration design space. We offer to the designer a way to improve performance and energy consumption on a high abstraction design level without considering any changes in the hardware and the functionality of the application.

The remainder of the paper is organized as follows. In Section 2, we provide an overview of the related work. In Section 3, we present our proposed dynamic data type refinement methodology. In Section 4, the case studies are introduced and the obtained simulation results are analyzed. Finally, in Section 5 we draw our conclusions.

2. Related Work

In general-purpose software and algorithms design [17], primitive data structures are commonly implemented as mapping tables. They are employed to accomplish software implementations with high performance or with low memory footprint. Additionally, the Standard Template C++ Library [13] provides many basic algorithms and data structures needed for implementing dynamic data structures in a general context. Data management and data optimizations for traditional (non-dynamic) embedded applications have been extensively studied in the related literature [1, 2, 15]. Also, from the methodology viewpoint, several approaches have been proposed to tackle this issue at the different levels of abstraction [2, 6, 16]. However, in modern dynamic applications the behavior of many algorithms is heavily determined by the input data. This means that multiple and completely different execution paths can be followed, leading to complex, dynamic data usage according to the behavior of the users. Therefore, our approach focuses on optimizing the DDTs for modern network application with runtime memory allocation needs, contrary to static compile-time data allocation optimizations.

In [3], various data structures and corresponding transformations are explored, applied in sequence to obtain low cost but extremely complex data structures and corresponding access operations. Additionally, more attention is paid to energy consumption and other embedded systems criteria in [5]. A fast, stepwise, cost-driven, and automated exploration and refinement were proposed in [8, 14] for multimedia applications at system level, which operate on large and irregular data structures that typically exist in this application domain. Contrary to our work, this exploration was tuned for multimedia applications. Multimedia applications have very different dynamic data access patterns (e.g. extensive data re-use) compared to networking applications. Therefore, more network-sensitive criteria have to be explored in order to arrive to optimal dynamic data types.

The work presented here is related to [18]. However, there are major differences with the work presented there. We focus on the data-structure optimizations in the context of network applications, extending the range of applications and without being limited to a specific heavy data-oriented network router. Secondly, we analyze the software implementation of energy efficient data structures as opposed to explicitly designing and using specifically configured physical memories. In our context, we assume that the embedded platform is already designed and that our dynamic data types, which are tuned to the network application, are incorporated in the middleware on top of the given platform hardware. Thirdly, we extend considerably the exploration of the dynamic data type design space. This extension is done by adding the factor of network-configuration to explore more consistently the search space. Our approach studies the optimizations and possible trade-offs related to additional design metrics that are key factors in embedded systems, such as low energy consumption and memory accesses, on top of the usual ones, namely performance and memory footprint. Finally, we support the whole methodology flow for the first time with fully automated tools.

3. DDT Refinement Methodology

The methodology enables the systematic refinement of dynamic data types for new network applications, implemented in embedded systems and consists of three distinct steps applied to each network application (Figure 1). More specifically the first step is the exploration of the DDTs at application level, where DDTs are refined according to the dynamic data access behavior of the network application

under study. The second one is the exploration of the DDTs at network configuration level, where DDTs are refined according to the configurations that the network application is going to use. Finally, the third step is the exploration of the DDTs at Pareto space level, where DDTs are refined according to the design constraints of the embedded system that they will be implemented on.

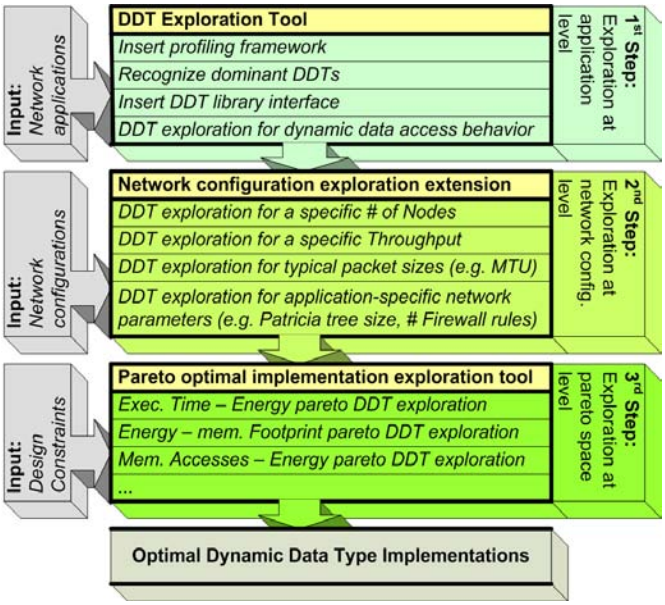

Figure 1. 3-step DDT Refinement methodology flow

The objectives of this exploration framework are:

- The refinement of the DDTs at three distinct steps. Each step adds one more refinement level achieving better experimental results each time we refine further.
- The automation of the whole exploration process, because it requires $100s$ or even $1000s$ of simulations.
- The use of a stepwise procedure propagating restrictions from one step to the next, in order to decrease the number of total simulations needed, thus reducing the total design time cost.

3.1. Application-level DDT Exploration

In the first step of the proposed methodology, we explore the DDTs at the application-level, in order to find the optimal DDT combinations for the dynamic data access behavior of the application under study. To achieve that we have to perform two substeps. First, we attach to each candidate DDT of the network application a profile object and run the application for some typical input traces. The profiling reveals the dominant data structures of the application (i.e. the ones that are accessed the most), which are going to be explored. Then, we insert, just once, inside the source code the instrumentation linking each dominant data structure of the network application with our C++ DDT library. The C++ DDT library is comprised of 10 different DDTs and developed in [9].

The instrumentation consists of typical functions operating on DDTs (e.g. add a record, access a record or remove a record). This procedure does not alter the actual functionality of the application. Then, the exploration is performed in an automatic way by keeping the same instrumentation and changing the DDT implementation for each dominant data structure. All DDTs in the C++ library (and combinations of them) are used in the exploration, simulated and profiled at run-time. The whole procedure takes from 0.8 up to 64 seconds per simulation for a single DDT combination according to the application. By using the term simulation we mean an execution of an application under study using as input a network trace.

We simulate all the combinations of DDTs for the chosen network application. For example, if there is one dominant data structure, then we have to simulate 10 times, one time for each different DDT. If there are two dominant data structures, then we have to simulate 100 times (i.e. 10 different DDTs for the first dominant data structure combined with 10 different DDTs for the second dominant data structure and so on). Then, we automatically keep the combinations, which have the lowest energy consumption, shortest execution time, lowest memory footprint and lower memory accesses. The energy estimations are calculated using an updated version of the CACTI model [12]. According to our experimental results (Section 4) approximately 80% of the DDT combinations produce not optimal results for all the aforementioned metrics. Thus, this procedure will discard approximately 80% of the available DDT combinations (Table 1).

3.2. Network-level DDT exploration

To automate this exploration step we have developed an extension to our DDT exploration framework. The first part of the tool (written in Perl) can recognize automatically the differences between the various network configuration implementations. This is done by parsing the available network traces and extracting the network parameters from the raw data in the traces. Then, the second part of the tool (written in C++) can automatically do the DDT implementation for all the different network configurations in the network application.

This is a critical step of the methodology, because our experimental results show that for different network configurations, the optimal DDTs vary greatly for certain metrics (Section 4). The network parameters, which are important for the DDT exploration, are: the number of nodes in the network, the throughput of the network and the typical packet sizes used (e.g. Maximum Transmission Unit

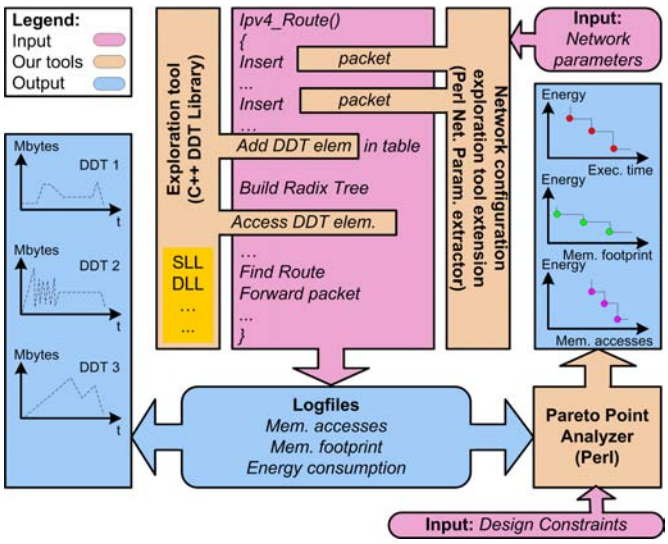

Figure 2. Tool support of the 3-step DDT Refinement methodology flow

packet size). Finally, other network parameters used for the DDT exploration are application specific. For example, the Radix tree size is an important parameter for the IPv4 routing application [10], affecting greatly the DDT exploration. Other examples are the Level of Fairness used in the Deficit Round Robin [10] scheduling application and the number of rules activated in a firewall application.

This exploration step requires input traces, which are typical of the network configuration. With the help of our tool we parse these input traces and we extract from them automatically the parameters of the network. In order to test the validity of our results we have used a total of 10 traces from 8 different networks. The first three networks come from the NLANR [11] and originate from the total campus and satellite buildings activity. The rest come from Dartmouth University's collection of wireless network traces [7] on the corresponding campus buildings.

More specifically, we take the remaining 20% DDT combinations of the previous step and simulate each one of them for all different network configurations. For example, this means that we will perform approximately 80 simulations, if we explore for four different network configurations in an application with two dominant data structures (i.e. 20% of 100 DDT combinations equals 20 DDT combinations. 20 DDT combinations $\times$ 4 different network configurations equals 80 simulations). If we had not reduced the exploration space at the previous step, then we would had to perform 400 simulations. Then, we automatically keep track of the combinations, which have the lowest energy consumption, shortest execution time, lowest memory footprint and lower memory accesses (Table 1).

3.3. Pareto-level DDT exploration

In the third step, we explore the DDTs at the Pareto point level. Hence, instead of giving the designer as solution a single DDT combination, we provide a Pareto-optimal set represented by a Pareto curve. Every point in the set is better than any other solution in at least one metric. Thus, design constraints can be implemented directly in the exploration approach and get the best tradeoffs from the final DDT implementation. In order to provide to the designer the Pareto curve, we have developed another tool (written in Perl), which processes the Gigabytes of the log files produced by previous steps, and represents graphically all the DDT exploration solutions (Figure 3). Then, the Pareto space of all the solutions is pruned and the tool produces graphically the Pareto curves for the memory accesses, execution time, energy consumption and memory footprint tradeoffs (e.g. three of them can be seen in Figure 4). The rest of the DDT combinations are discarded. Then, the designer can choose very easily between a set of application-tuned Pareto optimal DDT implementations, which are within the design constraints (see Table 1).

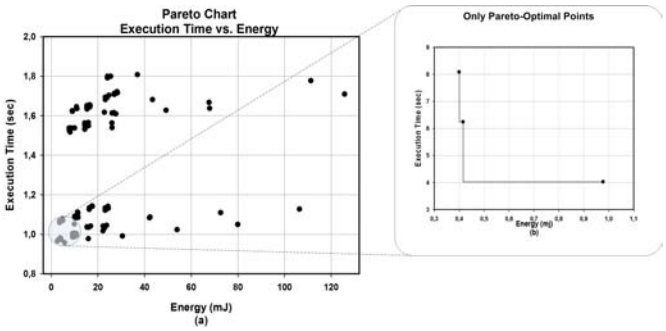

Figure 3. (a) Performance vs. Energy Pareto Space of URL (b) Pareto Optimal Points Graph

4. Experimental Results

We apply the proposed methodology to four realistic case studies representing different modern network applications selected from a broad variety: the first one is a routing application (Route), the second one is a context switching algorithm (URL), the third one is a firewall application (IPchains) and the fourth one is a Deficit Round Robin scheduling application (DRR). All applications are taken from the NetBench Benchmarking suite [10]. The results were obtained with gcc-3.3.3 on a Pentium4 1.6 GHz with 512 MB RAM running Linux kernel 2.6.4. All the results presented here are average values after a set of 10 simulations for each application, where all the final values were very similar (variations of less than 2%).

By using the proposed methodology the designer can achieve significant gains in the number of simulation com-

pared to an exhaustive exploration. The reduction achieved, in the aforementioned case studies is depicted in Table 1, with an average reduction of 80%.

Network applications	Exhaustive simulations	Reduced simulations	Pareto optimal
1. Route	1400	271	7
2. URL	500	110	4
3. IPchains	2100	546	6
4. DRR	500	60	3

Table 1. Reduction of total simulations needed to explore the design space

The first case study is the Route application [10]. The routing table uses a radix tree structure, which is the data structure to hold both host addresses and network addresses. In the first step, we determine the dominant DDTs of the application. Two dominant DDTs are present in the Route application, `radix_node` structure forms the nodes of the tree and the `rtentry` structure holding the route entries and contain other useful pointers. This means that we are going to simulate automatically a maximum of 100 combinations of DDTs to do exploration at application level.

In the second step, exploration is done at network configuration level. Seven network configurations were used, utilizing 7 different networks. Additionally, exploration was performed for 2 different values of network parameter Radix Tree (for 128 and 256 entries). The created log files contain detailed information concerning the DDTs' behavior: number of memory accesses, memory footprint, dissipated energy and execution time.

In the third step, the postprocessing tool parses the log files and produce the Pareto-optimal points for memory accesses–memory footprint and execution time–energy. Each network configuration is represented by a curve and each point the combination of DDTs along with the corresponding values concerning memory accesses, memory footprint, dissipated energy and execution time. Then, the Pareto curves are drawn (Figure 4), giving the designer a visual aspect of the available solutions. For example, in Figure 4a Pareto-optimal curve are depicted, considering a routing table size of 128 elements and for seven different networks. Assuming Radix Tree size 256 and Berry trace [7], Figure 4b shows the Pareto-optimal point, within the solid circle, is the combination of array and double linked list DDTs, with energy dissipation 6.4 mJ, execution time 0.17 sec., memory footprint 477, 329 Bytes and 4, 578, 103 accesses.

For comparison reasons, if double linked lists were used, the application would demand 68.8% more memory footprint, 12% more energy and 12.5% gains in execution time from the best Pareto-optimal point of each corresponding metric (shadow area of Figure 4b and Figure 4c). Tradeoffs can be achieved up to 90% for the dissipated energy, 20% for the execution time, 88% for the memory accesses and 30% for the memory footprint. Comparing these solutions with the remaining Pareto points, which do not belong to the Pareto-optimal curve, the gains become bigger. Particularly, we experience a reduction in memory accesses up to a factor of 8, for memory footprint up to a factor of 12, for dissipated energy up to a factor of 11 and for execution time up to a factor of 2.

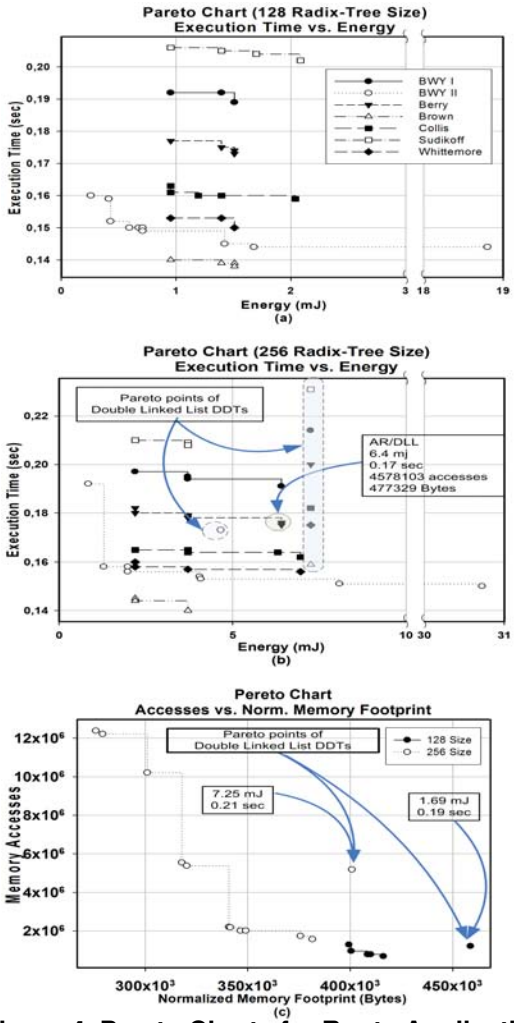

Figure 4. Pareto Charts for Route Application Execution Time vs. Energy (a) table size 128, (b) table size 256 (c) Accesses vs. Memory Footrpint (BWY I).

The second case study is URL-based switching [10]. We simulated the application a maximum of 500 times (100 different DDT combinations $\times 5$ different networks) to ex-

plore the DDTs at network configuration level. The simulation results were filtered to get Pareto-optimal points. In URL the best combination of DDTs in terms of energy gives a 52% reduction in comparison to the most energy-consuming Pareto-optimal point. This percentage is the average of the five traces. The corresponding reduction percentage of time is 13%, of memory footprint 70% and of memory accesses 82%. Other combinations can take up to double execution time to run, consume up to 30 times more energy, have up to 4 times more memory accesses and allocate up to 5 times more memory footprint comparing to the average of the Pareto-optimal. A comparison with the initial NetBench DDT implementations is worthwhile (both DDTs were implemented as single linked lists). The execution time is reduced by 20% and energy by 80%.

By applying the methodology to the other two application we achieved similar results. Due to lack of space the trade-offs among the Pareto-optimal points achieved, are presented in Table 2 for the four case studies.

Application	Energy	Exec. Time	Mem. Accesses	Mem. Footprint
1. Route	90%	20%	88%	30%
2. URL	52%	13%	70%	82%
3. IPchains	38%	3%	87%	63%
4. DRR	93%	48%	53%	80%

Table 2. Trade-offs achieved among Pareto-optimal points

5. Conclusions

We have presented a systematic approach to explore all possible implementations and combinations of DDTs, using a novel methodology and supporting automation framework, which has led to significant improvements in terms of energy consumption, execution time and memory footprint in combination with a reduce design time cost. This methodology shows that, the choice of an optimal implementation of a dynamic data type can be flexibly tuned to the specific needs of each application, each network configuration and each embedded system constraint. The design flow methodology was verified by exhaustive simulation under various conditions, traces and DDTs implementations. Furthermore, it was shown that we can reach, with the use of our methodology, energy savings 80% and increase in performance 22% (in average) of the original benchmarks implementation without any increase in memory footprint and memory accesses. Finally, trade-offs among the Pareto-optimal choices provide alternative solutions to the designer.

Acknowledgements

This work is partially supported by the E.C. funded program AMDREL IST-2001-34379, `http://vlsi.ee.duth.gr/amdrel`, the Greek Government PYTHAGORAS II and PENED 03ED593 Grant, Spanish Government Research Grant TIN 2005-5619 and E.C. Marie Curie Fellowship contract HPMT-CT-2000-00031.

References

[1] L. Benini and et al. Increasing energy efficiency of embedded systems by application-specific memory hierarchy generation. *IEEE Des. & Test*, 2000.

[2] F. Catthoor and et al. *Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design.* Kluwer Academic Pub., 1998.

[3] E. G. Daylight and et al. Memory-access-aware data structure transformations for embedded software with dynamic data accesses. *IEEE Tran. on VLSI Systems*, 12, 2004.

[4] T. Givargis and et al. System-level exploration for pareto-optimal configurations in parameterized systems-on-a-chip. In *Proc. of ICCAD*. IEEE Press, 2001.

[5] J. J. L. da Silva and et al. Efficient system exploration and synthesis of applications with dynamic data storage and intensive data transfer. In *Proc. of DAC*. ACM Press, 1998.

[6] M. Kandemir and et al. Dynamic management of scratch-pad memory space. In *Proc. of DAC*. ACM Press, 2001.

[7] D. Kotz and K. Essien. Analysis of a campus-wide wireless network. In *Proc. of MobiCom*. ACM Press, 2002.

[8] M. Leeman and et al. Methodology for refinement and optimisation of dynamic memory management for embedded systems in multimedia applications. In *Proc. of SiPS*, 2003.

[9] S. Mamagkakis and et al. Design of energy efficient wireless networks using dynamic data type refinement methodology. In *Proc. of WWIC*, 2004.

[10] G. Memik and et al. Netbench: A benchmarking suite for network processors. In *Proc. of ICCAD*. IEEE Press, 2001.

[11] NLANR. `http://www.nlanr.net`.

[12] A. Papanikolaou and et al. Global interconnect trade-off for technology over memory modules to application level: case study. In *Proc. of SLIP*. ACM Press, 2003.

[13] P. Plauger and et al. *The Standard Template Library.* Prentice-Hall, 1998.

[14] L. Séméria and et al. Resolution of dynamic memory allocation and pointers for the behavioral synthesis form c. In *Proc. of DATE*. ACM Press, 2000.

[15] S. Steinke and et al. Assigning program and data objects to scratchpad for energy reduction. In *Proc. of DATE*. IEEE Computer Society, 2002.

[16] M. Verma and et al. Dynamic overlay of scratchpad memory for energy minimization. In *Proc. of CODES+ISSS*. IEEE Computer Society, 2004.

[17] D. Wood. *Data Structures, Algorithms, and Performance.* Addison-Wesley Longman Publishing Co., Inc., 1993.

[18] C. Ykman-Couvreur and et al. Exploration and synthesis of dynamic data sets in telecom network applications. In *Proc. of ISSS*. IEEE Computer Society, 1999.