Design of Energy Efficient Wireless Networks Using Dynamic Data Type Refinement Methodology

Stylianos Mamagkakis[1], Alexandros Mpartzas[1], Georgios Pouiklis[1], David Atienza[3], Francky Catthoor[2,*], Dimitrios Soudris[1], Jose Manuel Mendias[3], and Antonios Thanailakis[1]

[1]VLSI Design and Testing Center-Democritus University, Thrace, 67100 Xanthi, Greece
{smamagka, ampartza, gpouikli, dsoudris, thanail}@ee.duth.gr
[2]IMEC vzw, Kapeldreef 75, 3001 Heverlee, Belgium
Francky.Catthoor@imec.be
[3]DACYA UCM, Avda Compute nce s/n, 28040, Madrid, Spain
{datienza, mendias}@dacya.ucm.es

Abstract. This paper presents a new perspective to the design of wireless networks using the proposed dynamic data type refinement methodology. In the forthcoming years, new portable devices will execute wireless network applications with extensive computational demands (2 – 30 GOPS) with low energy consumption demands (0.3 – 2 Watts). Nowadays, in such dynamic applications the dynamic memory subsystem is one of the main sources of energy consumption and it can heavily affect the performance of the whole system, if it is not properly managed. The main objective is to arrive at energy efficient realizations of the dominant dynamic data types of this dynamic memory subsystem. The simulation results in real case studies show that our methodology reduces energy consumption 50% on average.

1 Introduction

Wireless communications have experienced a rapid growth over the latest years. The complexity of modern wireless networks is increasing, supporting a wide variety of services. Such complex systems require a combination of hardware and embedded software components in order to deliver the required functionalities at the desired performance level. Additionally, portable computers like PDAs and laptops using this wireless communication to interact with the environment rely on their limited battery energy for their operation. Energy consumption is the limiting factor in the amount of functionality that can be placed in these devices. More extensive and continuous use of network services by multimedia applications will only aggravate this problem.

Network applications are characterized by their various input and output streams, having different quality of service requirements. Depending on the service class and QoS of a connection, the memory footprint and accesses of these applications vary greatly. Therefore, the use of dynamic memory is imperative and must be in accordance to the dynamic behavior of the application. This behavior is often characterized

* Also professor at the Katholieke Univ. Leuven, Belgium

P. Langendoerfer et al. (Eds.): WWIC 2004, LNCS 2957, pp. 26–37, 2004.

by complex algorithms that operate on large dynamically allocated stored data structures (i.e. single and double linked lists, arrays, dynamic first-in-first-out buffers) and are usually implemented with the use of the C or C++ programming language. The data are used for communication between multiple processes and may be shared among concurrent tasks. Adaptations to the dynamic nature of wireless networks are necessary to achieve energy efficiency and acceptable QoS.

The wireless network applications considered in this paper are encountered in the middle layer protocol processing and can be found from the MAC layer up to the transport layer of the OSI protocol stack. These are algorithms operating on large and irregular data structures, which are allocated and stored dynamically as dynamic queues, lists and association tables of records indexed by multiple keys, with the support of services to insert, locate, remove or substitute a record. Finally, stringent real-time requirements apply due to the very high bit-rate I/O data streams [1].

In this paper, we present a new methodology that allows developers to design wireless network applications with reduced memory energy consumption by utilizing the different dynamic data types (DDTs from now on) available in the construction of the dynamic memory subsystem. First, we define the relevant DDT search space, including the various DDTs, their combinations and their multilayered implementation. Then, we propose a way to find the dominant DDTs of the wireless network applications, explore all the available options in the search space and conclude to refined DDTs that will consume the least energy possible.

The remainder of the paper is organized as follows. In Section 2, we describe some related work. In Section 3, we define the search space of DDTs and some wireless network application behaviors. In Section 4, we present the DDT refinement methodology. In Section 5, we introduce our case studies and present the simulation results obtained. Finally, in Section 6 we draw our conclusions.

2 Related Work

The work presented in this article is inspired by [2], [3] and [4]. There are however three main differences. First of all, as opposed to optimizing heavy data-oriented multimedia applications (e.g. 3D Games, 3D rendering algorithms etc. [4]), this article focuses on wireless network applications. We show that by applying similar DDT refinements, wireless network applications consume significantly less energy.

The second difference is the software implementation of energy efficient DDTs as opposed to explicitly designing and using specific, configured, physical memories (hardware) [3]. We assume that the hardware is already designed and fully functional and that the refinement of the software will lower the memory energy consumption.

The third difference with the referenced work is that instead of focusing on the table lookups and on the accesses of the table keys [2], we explore all the DDTs of the network applications and the available search space in a more detailed way.

Optimizations and techniques for general purpose design to reduce energy consumption are explained in [5]. However, refining the DDTs at the software level in wireless network applications with complex dynamic behavior has not been given

much attention. Related work, on wireless network protocols [16] is supplementary to our work and supports our results.

In this paper, we propose using a fast, stepwise, cost-driven exploration and refinement for the DDTs in wireless network applications at the highest abstraction level, where the impact on memory performance and consumption is the most crucial.

3 Dynamic Data Type Search Space and Application Behaviors

3.1 Dynamic Data Type Search Space

In this subsection we introduce the DDTs available for our exploration and final refinement. These DDTs consist of various sets of data (records) put together, usually in the form of doubly linked lists [6]. They differ in the way these data types are interconnected and in the way that records can be added or subtracted during runtime to adjust themselves to the dynamic nature of the application. The DDT search space consists of the basic DDTs, their combinations and variations (as shown in Table 1).

Table 1. Abbreviations used throughout the text

Abbreviation	Explanation
CLS1WEL	"Embedded" single linked list of arrays
CLS1WPLO	"Pointer" single linked list of arrays with roving pointer
CLS1WPL	"Pointer" single linked list of arrays
CLS2WEL	"Embedded" double linked list of arrays
CLS2WPLO	"Pointer" double linked list of arrays with roving pointer
CLS2WPL	"Pointer" double linked list of arrays
CLSAR	Simple array
CLSPA	A pointer array of arrays
LL1WEL	"Embedded" single linked list
LL1WPLO	"Pointer" single linked list with roving pointer
LL1WPL	"Pointer" single linked list
LL2WEL	"Embedded" double linked list
LL2WPLO	"Pointer" double linked list with roving pointer
LL2WPL	"Pointer" double linked list

The basic dynamic data types are:

- **Array (CLSAR).** An array is a set of sequentially indexed elements having the same intrinsic data type, which is called element of the array. Each element of the array usually is a record of the application. All arrays consist of contiguous memory locations. The average access count to a random element is 1.

- **Single linked list (LL1WEL).** A single linked list is a set of data types or data structures that are connected with each other via pointers. Each element of the single linked list holds a record of the application and points to the memory address of the next element. The average access count to a random element is (*N/2 + 1*), where *N* is the number of total elements.
- **Double linked list (LL2WEL).** A double linked list is a set of data types or data structures that are connected with each other via pointers. Each element of the double linked list holds a record of the application and points to the memory address of the previous and the next element. The average access count to a random element is (*N/4 + 1*), where *N* is the number of total elements.

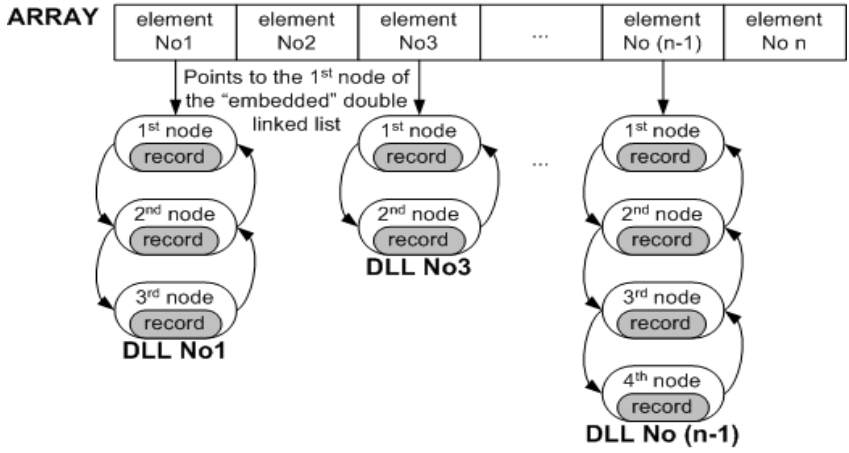

Fig. 1. An array of double linked lists with embedded records

These basic dynamic data types can have variations:

- **Embedded (EL).** In the embedded variation the record of the application is stored within the DDT. This means that the DDTs must be stored in bigger memories (to include memory space for the records) but no additional memory accesses are needed to read the records. An example of "embedded" records is shown in Fig. 1.
- **Pointer (PL).** In the pointer variation the record of the application is stored outside the DDT and is accessed with a pointer. This means that the DDTs can be stored in smaller memories but the records will need one more extra memory access (for the pointer) to read them. An example of "pointer" records is shown in Fig. 2.
- **Roving pointer (O).** The roving pointer is an auxiliary pointer to access a particular element of a list with less memory accesses [7]. If an application accesses its records sequentially, we can store the address of the element, which was accessed last, in a pointer and use it as a start for the next element access. This way, the access count to an element can be reduced drastically from half the list size down to two memory accesses.

Finally, the basic data types can be combined. In this way, we can have arrays of "embedded" doubly linked lists (as shown in Fig. 1), single linked lists of "pointer"

single linked lists (as shown in Fig.2), doubly linked lists of "embedded" arrays, arrays of "pointer" single linked lists with roving pointer etc. The difference between these complex dynamic data structures lie within the memory footprint and the memory accesses needed to access a random element.

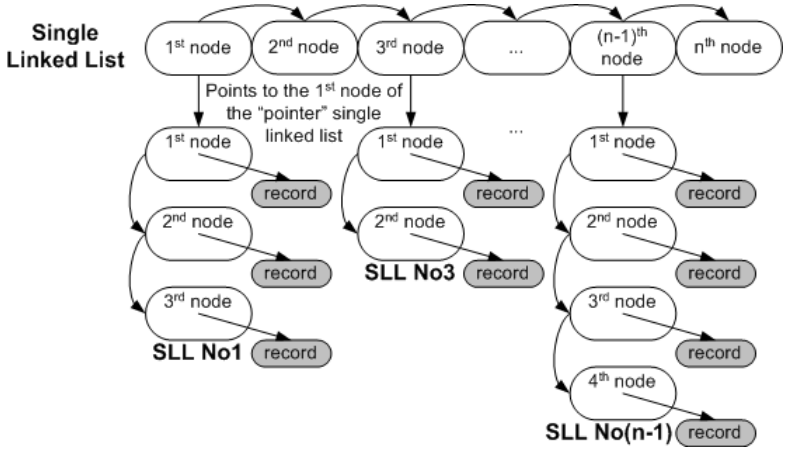

Fig. 2. A single linked list of single linked lists with "pointer" records

For example if we have 100 records, we can split them by using 2 keys instead of 1 and have 20 sets of 5 records. This means that we are transforming a single linked list (with 100 elements) to a single linked list (with 20 elements) of single linked lists (with 5 elements). So instead of having an average access count of 50 for a random element, we have now an access count of 12.5. The tradeoff comes of course with memory footprint (instead of allocating memory space for a simple 100-element data structure, we allocate memory space for a complex 120-element data structure).

An example of access count improvement on various two layered dynamic data structures is shown in Table 2, whereas *N* is the total number of elements of the list and *A* is the total number of elements of the array.

Table 2. Access count example to a random element of various DDTs

Dynamic data type	Access count to a random element
CLS1WEL	*N/2A + 2*
CLS1WPLO	*N/2A + 3A/N +1*
CLS1WPL	*N/2A + 3*
CLS2WEL	*N/4A + 2*
CLS2WPLO	*N/4A + A/5N + 5/4*
CLS2WPL	*N/4A + 3*
CLSAR	*1*
CLSPA	*2*
LL1WEL	*N/2 +1*
LL1WPLO	*N/2 + 1/N*
LL1WPL	*N/2 + 2*
LL2WEL	*N/4 + 1*
LL2WPLO	*N/4 + 2/N + ¼*
LL2WPL	*N/4 +2*

In a similar fashion to 2-layered implementations, n-layered implementations of DDTs can be constructed. A common implementation of multi-layered DDTs is the "tree". A tree is a pointer array of pointer arrays of pointer arrays etc. Each level of the tree is in reality a layer of pointer arrays, and the final level is where the records actually reside. In this paper, only single and two layered implementations will be considered, but this work can be easily extended to multilayered implementations.

3.2 Application Behaviors

Four different kinds of application dynamic behavior exist [8]. Namely, look up, iterate, insert and remove. Look up behavior corresponds to the retrieval of a data element, given a specific key value (e.g. source or destination IP address). Iteration behavior corresponds to the traversal of all the data elements that are stored in the DDT regardless of the associated key value of every data element. Insert behavior corresponds to the insertion of data elements. And finally, remove behavior corresponds to the removal of data elements.

Depending on the software design, the application under investigation may be look up dominant, iteration dominant, insert dominant, remove dominant or combinations of these. In all cases, we assume that look up, insert and remove behavior have to take place in a constant amount of time. The execution times of these operations may thus not depend on the total number of stored data element in the DDT.

Therefore the amount of total memory accesses does not depend only on the DDT selected but also on the behavior of the application. To be more precise, different behaviors favor in terms of energy consumption some DDTs more than others.

4 Dynamic Data Type Refinement Methodology

4.1 Cost Function

Wireless network applications must run on mobile devices. These devices are realized with the use of state-of-the-art embedded systems. In these embedded systems, the performance of the aforementioned applications is considered a hard constraint to meet, whereas area and energy are cost factors and must be optimized. This paper focuses on the optimization of the energy consumption factor.

The embedded systems, in which wireless network applications are realized, consist of on-chip and off-chip memories. During normal communication conditions of wireless devices [14], the main source of energy consumption is the data transfer and storage in these memories [9]. For on-chip memories, the energy consumption of one memory access increases with the memory size, while for off-chip memories, it can be considered more or less independent from the memory size, and a significant portion goes into the off-chip and communication. Hence energy can be saved either by reducing the number of memory accesses, or by storing data into smaller on-chip memories, or by doing both. Also, note that the relation between the memory accesses or the memory size and the energy consumption is not linear.

To this end, the refinement of DDTs, which make most of the data transfer and storage, should mainly aim at minimizing the memory footprint and data accesses to achieve the desired energy consumption reduction.

Finally, to estimate the consumed energy by the wireless application through the use of the various DDTs, we have used an updated version of the CACTI model [10]. This is a complete energy/delay/area model for embedded SRAMs that depends on memory footprint factors (e.g. size, internal structure or leaks) and factors originated by memory accesses (e.g. number of read or write accesses and technology used).

4.2 Refinement Methodology

As mentioned in Section 1, the dynamic memory subsystem can have important influence on the overall application performance. Our design method focuses on a high level specification of the network applications and on profiling tools that are used to extract information about the important factors concerning the memory performance, namely memory size, memory accesses and memory power consumption. As a result, repeated refinements are done and estimations from these refinements are used to define the DDT that consumes the least power. It must be noted that the functionality of the application is not changed at all during this process.

The whole process is divided in three main steps. First, the method analyzes the access pattern of the DDTs involved in the application and optimizes their implementations preserving the hard constraints set by the wireless network application. This is done with the use of the Matisse tool [4]. Secondly, global dynamic memory managers are considered in the design flow to tackle the allocation and de-allocation of the DDTs. Finally, the last step is the physical memory management which deals with the allocation of the DDTs in specific memories or memory hierarchies.

The dynamic and physical memory managers are not detailed yet, because we want to concentrate on a pure software implementation and our aim is not a specific embedded platform. Therefore, only the DDT refinement phase will be explained in the following steps:

1. **Common interface installation.** Firstly, we have to implement a certain set of basic operations about the handling of elements to provide a common interface between any DDT and the application. In this way, no matter which DDT we use, we will not have to change the application source code over and over again.
2. **Profiling tools installation.** Secondly, we are going to insert the Matisse profiling tools in every DDT, so that we can get measurements about the usage of each DDT and thus decide, which are the dominant ones that need refinement. The profiling tools can be easily embedded in any DDT with the use of a Perl script, provided with the Matisse tool. This is a semi-automatic process and can be easily applied to large programs.
3. **Search space exploration.** Thirdly, we explore the available DDT search space by implementing each DDT in the source code of the wireless network application. The memory accesses, the normalized memory footprint and energy consumption of all these DDTs are measured by the profiling tools and stored in huge log files (the normalized memory footprint is the average size occupied by each DDT). The

Matisse tool contains a C++ library of 14 single and two-layered DDTs that can be easily implemented and profiled.

4. **MATLAB calculations.** Fourthly, with the use of a minimization-problem function developed in MATLAB, we get the final values about the energy consumption of each DDT.
5. **Common interface and profiling tools removal.** Finally, we remove the "basic operations interface" and the profiling tools and implement the DDT that we have selected, which is the most energy efficient.

Due to the dynamic nature of the wireless network applications, special care must be taken in the profiling of the system. Typical trace inputs and behavior of the specific wireless network, which is designed, must be considered. For example, there are going to be differences about the most energy efficient DDT implementation between a Wireless Local Area Network (WLAN) and a Wireless Personal Area Network (WPAN). Therefore, it is common in the simulation of a wireless network, with low internet traffic, to need a very simple data type to achieve the least power consumption. On the other hand, high internet traffic wireless networks need more complex DDTs with multilayer implementations. In any case, our methodology steps must be followed to arrive in the most power efficient solution.

5 Case Studies and Simulation Results

We have applied the proposed methodology in two case studies from the network application domain: the first case study is Route [12], an algorithm that implements IPv4 routing according to RFC 1812 and the second one is DRR [13], a scheduling algorithm. The methodology can be used both to the wired and wireless domain [14][15].

In the following subsections we describe briefly the behavior of the two case studies and the proposed approach is applied to obtain profiling values (e.g. energy consumption, memory usage and memory accesses). The results have been obtained with the `gcc`-2.95.3 on a Pentium II at 350 MHz with 132 Mbytes SDRAM and running FreeBSD 4.8-Release 3. All the results are average values after a set of 10 simulations for each application and DDT implementation, where all the final values are very similar (variations of less than 2%). Finally, the design time needed to achieve these refinements has been one week for each application.

5.1 Method Applied to a Routing Application

The first case study presented is the Route application, which is taken from the NetBench benchmarking suite [11]. The application implements the table lookup along with internet checksum for the header. It makes the necessary changes to the header (e.g. to the Time-To-Live value), fragments the packet if necessary and forwards it. The source code is taken from the FreeBSD Operating System [12].

A "route" is a defined pair of addresses: a destination and a gateway. The pair indicates that if you are trying to get to this destination, communicate through this gateway. The routing table is implemented by the radix algorithm, which is a member of the Route application. Basically, the radix algorithm builds and maintains radix trees for routing lookups. A tree is being built with internal nodes and leaves. The leaves

represent address classes, and contain information common to all possible destinations in each class. As such, there will be at least one mask and prototype address. Each internal node represents a bit position to test.

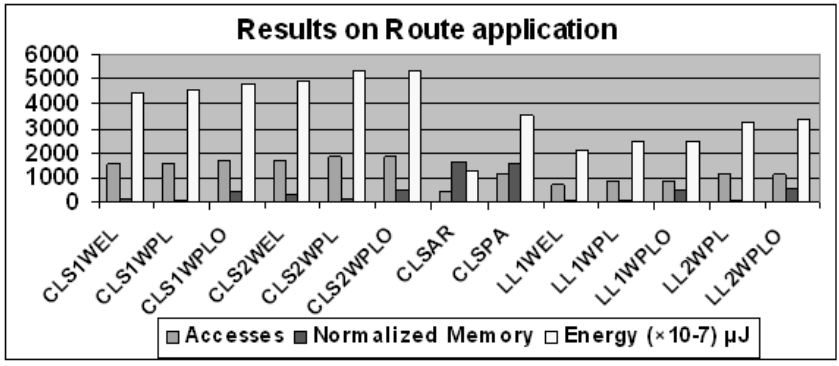

Fig. 3. Profiling values for different dynamic data types in the Route application

The data structure for the keys is a radix tree with one way branching removed. The index `rn_b` at an internal node n represents a bit position to be tested. The tree is arranged so that all descendants of a node *n* have keys whose bits all agree up to position `rn_b – 1`. There is at least one descendant which has a bit with value '1' at position `rn_b`, and at least one with the value '0' there. A route is determined by a pair of key and mask. The mask that we use has the value 255.255.255.255. The source code makes use of normal routes with short-circuiting an explicit mask and compare operation when testing whether a key satisfies a normal route, and also with remembering the unique leaf that governs a sub tree.

The routing application has been profiled in our results for an input trace of 20,000 packets. The size of the packets is varying from 0 to 512 bytes. This means that the memory needed to store the data is not fixed but it depends on the size of the incoming packet. That illustrates the dynamic nature of the Route application.

The next task to be performed is to find into the application's source code the dominant DDTs. In our case the dominant ones are the classes `radix_node` and `rtentry`. The class `radix_node` describes the structure of the routing table, which is implemented as a radix tree, and the class `rtentry` holds the routes that have been inserted into the routing table. After the dominant DDTs have been defined, we have used the Matisse tool in order to explore the DDT search space.

Comparing the estimates provided by the Matisse tool, conclusions can be easily drawn on which DDT should be used to achieve an energy efficient implementation of the application. Specifically, concerning the routing algorithm it has been found that the DDT that leads to an optimal implementation in terms of energy consumption is CLSAR. For instance if it is required to add or delete a node to the n^{th} level of the tree, with the CLSAR implementation only one access is needed to get to the desired position, whereas using the single list implementation we should traverse the tree down to the n^{th} level and then perform the desired action. Although this DDT imple-

mentation (CLSAR) gives a 53.4% increase in terms of normalized memory footprint, it also gives a 60.9% reduction in the energy consumed and a 63.4% reduction in memory accesses in comparison to the worst solutions, the DDTs CLS2WPL and CLS2WPLO (as shown in Fig. 3). Finally, a 78,5% reduction in execution time is observed using our methodology.

5.2 Method Applied to a Scheduling Application

The second case study presented is Deficit Round Robin (DRR from now on) fair scheduling algorithm that is commonly used for bandwidth scheduling on network links [13]. The algorithm is implemented in various switches currently available (e.g., Cisco 12000 series) plus it is part of the Netbench suite [11], which consists of a set of characteristic, common networking applications. Its format categorization is queue maintenance and packet scheduling for fair resource utilization.

In the DRR algorithm, the scheduler visits each internal non-empty queue, increments the variable deficit by the value quantum and determines the number of bytes in the packet at the head of the queue. If the variable deficit is less than the size of the packet at the head of the queue, then the scheduler moves on to service the next queue. If the size of the packet at the head of the queue is less than or equal to the variable deficit, then the variable deficit is reduced by the number of bytes in the packet and the packet is transmitted on the output port. The scheduler continues this process, starting from the first queue each time a packet is transmitted. If a queue has no more packets, it is destroyed. The arriving packets are queued to the appropriate node dynamically allocating memory for them and if no such exists then it is created.

In the simulation of the algorithm, 5,000 packets are served with sizes varying from 1,500 to 15,000 bytes, while ten source and destination addresses are used. The reason of these limitations is the memory limitations of the system on which the simulation run. Still they have no effect on the proposed methodology. It should be also noted that only the size of the packet data is of importance as the algorithm does not process the data themselves.

Two dominant DDTs are presented in DRR: the first is the class `Packets` (used to create the packets to be scheduled in queues), while the second is the class `Deficit_node` (used to create the queues in which the packets are scheduled). Even between these two it can be identified that the most important DDT is holding the packets of each queue since there is one `Packets` data type for every queue, while there is only one queue data type throughout the program. Furthermore, the elements of the `Packets` can have a size varying from 1,500 to 15,000 bytes (depending on the packet size), while on the other hand each element of the queue DDT has a 16 bytes size.

Comparing the estimates provided by the Matisse tool, conclusions can be drawn on which DDT should be used to achieve an energy efficient implementation of the application. Specifically, concerning the DRR algorithm it has been found that the DDT that leads to a minimal energy consumption is CLSAR. This type of dynamic data gives a 44.3% reduction in the energy consumed (as shown in Fig. 4) and a 69.3% reduction in execution time, in comparison to the most common implementation, which is a single linked list.

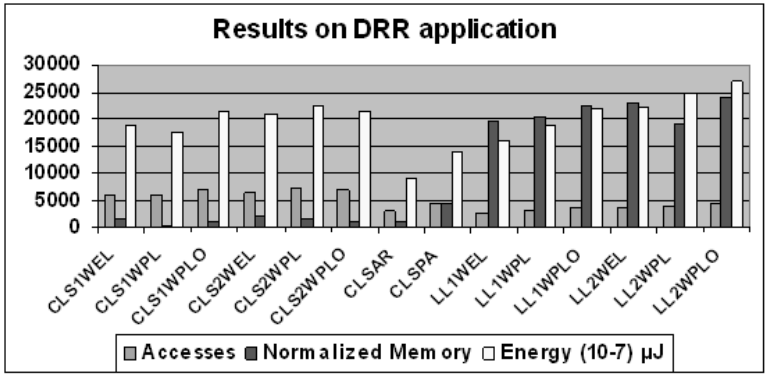

Fig. 4. Profiling values for different dynamic data types in the Route application

In order to give a notion of the spectrum of choices it should be mentioned that the worst solution concerning energy, consumes three times the amount of energy used by the optimal solution, making clear that the dynamic memory subsystem cannot be ignored when designing with low energy constraints in mind

In retrospect, we can see why an array serves our simulation better than the single linked list, a choice not so obvious for the programmer, initially. First of all, when trying to add a queue or a packet to a queue, the application does not have to traverse all the elements until it finds a null pointer (as it did with the linked list), to decide where to add the new element. Instead it can add it directly to the end of the array. The gains are even larger, when the look-up procedure does not examine the nodes one by one sequentially, but accesses to random nodes are made during runtime, because in this case the array only needs one access to get to the node, while the list needs n accesses to get to the n^{th} node.

6 Conclusions

As wireless networks grow in size and complexity, more complex wireless network applications with big dynamic memory requirements are employed to support them. In this paper, we prove the effectiveness of the proposed DDT refinement methodology to optimize the dynamic memory subsystem for the aforementioned applications.

This methodology allows a structured analysis and profiling of the memory access patterns hidden in algorithms with complex dynamic memory use. The way in which the data is stored for the studied algorithms is optimized with the use of refined DDTs. By doing this, the access patterns are also transformed and optimized, thus reducing the energy consumption of the wireless network application. Finally, the design is done in an easy step-wise method that gives the flexibility to the designer to meet the hard constraints of the application's performance and take into account the application's memory footprint as well.

Acknowledgements. Many people have contributed to the elaboration of this methodology and the development of the Matisse tool. We would like to especially thank Marc Leeman and Chantal Ykman-Couvreur from IMEC, Leuven. This work is partially supported by the European founded program AMDREL IST-2001-34379 and the Spanish Government Research Grant TIC2002/0750.

References

[1] K. Keutzer et al. "Mescal Project", http://www.gigascale.org/mescal/index.html, 2002
[2] S.Wuytack, F.Catthoor, H. De Man, "Transforming Set Data Types to Power Optimal Data Structures", *Proc. IEEE Intnl. Workshop on Low Power Design, Laguna Beach CA*, 1995
[3] C.Ykman-Couvreur, J.Lambrecht, D.Verkest, F.Catthoor, H.De Man, "Exploration and Synthesis of Dynamic Data Sets in Telecom Network Applications", *Proc. 12th ACM/IEEE Intnl. Symp. on System-Level Synthesis (ISSS), San Jose CA,* pp.125–130, December 1999
[4] M. Leeman, D. Atienza, C. Ykman-Couvreur, F. Catthoor, J. M. Mendias, "Methodology for Refinement and Optimization of Dynamic Memory Management for Embedded Systems in Multimedia Applications", *IEEE Intnl Workshop on Signal Processing Systems,* 2003
[5] L. Benini et al. "System level power optimization techniques and tools", in *ACM Transaction on Design Automation for Embedded Systems (TODAES),* April 2000
[6] S. Mamagkakis, M. Dasygenis, D. Soudris, and C. Goutis, "Data types, control and data flow structures of telecom network applications", *EASY Project IST-2000-30093*, February 2002
[7] Wilson, Johnstone et al. "Dynamic Storage Allocation, A survey and critical review", *Internation Workshop on Memory Management, Kincross, Scotland, UK,* 1995
[8] E. G. Daylight, T. Fermentel, C. Yckman-Couvreur, F. Catthoor, "Incorporating Energy Efficient Data Structures into Modular Software Implementations for Internet Based Embedded Systems", *ACM Intnl. Workshop on Software and Performance,* 2002
[9] F. Catthoor, S. Wuytack et al.,"Custom Memory Management Methodology – Exploration of Memory Organization for Multimedia System Design", *Kluwer Academic Publishers*, 1998
[10] N. Jouppi, "CACTI Model", http://research.compaq.com/ wrl/people/jouppi/CACTI.html
[11] G. Memik, B. Mangione-Smith, and W. Hu, "Netbench: A benchmarking suite for network processors", *CARES Technical Report*, 2001
[12] The FreeBSD Project, "FreeBSD Operating System", http://www.freebsf.org, 2003
[13] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round robin", *in Intnl. Proceedings of SIGCOMM,* Cambridge, MA, September 1995.
[14] P.J.M. Havinga, G.J.M. Smit, and M. Bos, "Energy efficient adaptive wireless network design", *Proc. 5 th Symposium on Computers & Communications (ISCC00)*, Antibes, France, July 3–7, 2000.
[15] K. Aida, A. Takefusa et al,. "Performance Evaluation Model for Scheduling in a Global Computing System", *Intnl Journal of High Performance Applications*, Vol 14 (No3), 2000.
[16] G. Dimitroulakos, A. Milidonis, M. Galanis, G. Theodoridis, C. Goutis, F. Catthoor, "Power Aware Data Type Refinement On The Hiperlan/2", *International Conference on Electronics, Circuits and Systems,* United Arab Emirates, 2003