Exploration methodology of dynamic data structures in multimedia and network applications for embedded platforms

Lazaros Papadopoulos*, Christos Baloukas, Dimitrios Soudris

VLSI Design and Testing Center, Department of Electrical and Computer Engineering, Democritus University Thrace, 67100 Xanthi, Greece

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Dynamic data structures
Embedded systems
Low energy consumption

ABSTRACT

In the last years, there is a trend towards network and multimedia applications to be implemented in portable devices. These applications usually contain complex dynamic data structures. The appropriate selection of the dynamic data type (DDT) combination of an application affects the performance and the energy consumption of the whole system. Thus, DDT exploration methodology is used to perform trade-offs between design factors, such as performance and energy consumption. In this paper we provide a new approach to the DDT exploration procedure, based on a new library of DDTs which remedies the limitations of an existing solution and allows the DDT optimization of a wider range of application domains. Using the new library, we performed DDT exploration in network and multimedia benchmarks and achieved performance and energy consumption improvements up to 22% and 5.8%, respectively.

© 2008 Elsevier B.V. All rights reserved.

1. Motivation

The emerging market of new embedded devices integrates multiple services such as multimedia and wireless network communications. They come from general purpose domain and the porting process to an extremely compact device is a difficult task. Additionally, such a kind of applications requires increased interaction with the environment, which raises their dynamism [1]. Multiple algorithms that need to run concurrently on such devices impose complex memory access patterns that may result in performance losses and high energy consumption.

There are many factors that contribute to the dynamic behavior of the network and multimedia applications. As far as the network applications are concerned, the varying size and timing of the packets that run the internet, create an unknown set of requests in data storage and data access. Also, in multimedia applications like video games, the unpredicted behavior of the user causes varying inputs. For example, the number of objects rendered on the screen can significantly vary and this leads to memory footprint variations and different access patterns that are unknown until runtime.

Every dynamic application keeps its data into structured entities called dynamic data types (DDTs), data structures or simply containers, like arrays, lists or trees. These data structures are responsible for keeping and organizing the data in memory and also servicing the application’s requests at runtime. The latter include instructions for storing, retrieving or altering a data value. The implementation of these operations depends on the chosen DDT and each one of them can favor specific access patterns. For instance, array-like DDTs offer quick retrieval of arbitrary data, but take up a lot of space, in contrast with lists that use exactly as memory space as needed, but are not suitable for arbitrary retrieve operations. Each application can host a number of DDTs and each one of them can be different according to its role in the program.

The above trade-offs should be taken into account, when designing the application. Choosing an improper DDT, may have negative impacts at the total amount of memory needed to execute the program. It will also affect the number of total accesses to the memory, thus consuming valuable resources from the embedded device that can cause performance issues and high energy consumption. Therefore, a systematic exploration that will help the designer to select the optimal data structure implementation for each application given the design constraints is needed. A DDT exploration methodology for selecting the optimal DDT implementation for a specific application is presented in [3]. This methodology allows the designer to make trade-offs between performance and energy consumption by selecting different DDT combinations. Thus, high improvements in performance and energy consumption can be achieved without considering hardware and application changes.

Data storage and access optimizations for static (non-dynamic) embedded applications have been extensively studied in the related literature [1,4,8]. Also, from the methodology point of view,
several approaches have been proposed to cope with this kind of applications at different levels of abstraction (e.g. memory hierarchies), such as the data transfer and storage exploration (DTSE) methodology [2]. However, in modern dynamic applications these solutions cannot be used, as they focus on static compile-time data allocation optimizations.

In [5] energy-friendly data structure transformations are performed. Both platform-dependent and platform independent transformations are briefly covered. In the aforementioned work the practical feasibility of the optimizations for a specific computer game scenario is demonstrated. In [6] various data structures and corresponding transformations are explored, applied in sequence and corresponding access operations. Additionally, automated exploration and refinement is proposed in [7,9] for multimedia applications at system level, which operate on large and irregular data structures that typically exist in this application domain. In contrast to this work, our approach is independent of the application domain and provides richer exploration space.

The work presented in this paper is more related to [3]. The authors provided a DDT exploration methodology in the network domain. The exploration procedure was supported by a library of DDTs, called Matisse profiling tool. It was an automatic and systematic methodology for optimizing dynamic applications. However, the methodology and the corresponding tool have several limitations, such as low flexibility and limited DDT support that do not allow the implementation of the methodology to complex applications with complicated dynamic behavior. In this paper, we introduce a DDT exploration methodology based on a new DDT library tool. The new library is designed following a completely different approach, introducing the concept of abstract data types (ADTs), supporting complex DDTs and providing a GUI environment. With the new DDT library, the DDT exploration is easier and can be applied in a wide range of application domains. We applied the methodology to network and multimedia benchmarks and achieved performance and energy consumption improvements up to 22%.

The paper is organized as follows: The new DDT library is presented in Section 2, while in Section 3 we describe the methodology for optimal DDT exploration using the new library. In Section 4 we present the experimental results for multimedia and network applications and evaluate the effectiveness of the new library. Finally, in Section 5 we draw our conclusions.

2. New library description

Table 1 shows the main differences between Matisse profiling tool and the new DDT library we developed. This section analyzes these differences and provides a brief description of the new library.

### 2.1. Matisse profiling tool

Matisse profiling tool used in [3] exhibits a number of critical limitations that did not allow the efficient DDT exploration of modern dynamic applications. The Matisse profiling tool did not distinguish between the DDT functionality and DDT implementation. Also, it did not exploit the advantages provided by the object oriented design. Therefore, the extension of its DDT library is hindered and the combination of DDTs in more complex structures is also difficult.

An important limitation of the Matisse profiling tool is that it consisted of a relatively small number of DDTs. However, modern multimedia and network applications contain complex data structures, such as sorted lists, trees, hash tables, etc. Therefore, it was not possible to apply DDT exploration in these kinds of applications.

Additionally, limitations in the design of Matisse profiling tool did not allow the easy integration with the applications, because each object was treated as an array of basic data types, (for example integers), resulting in increased complexity. Therefore, Matisse could only be applied to relatively small applications of limited complexity. For larger applications, with many DDTs, applying the Matisse interface is infeasible and a very time consuming process. The contemporary complex applications require a more flexible approach.

In case of having multiple DDTs used throughout the application under exploration, evaluation of all possible combinations of DDTs is required for specifying the optimal one. However, the user may decide to evaluate certain combinations, so as to reduce the exploration space and therefore, reduce the exploration time (although in this case the exploration is not full). Matisse did not provide this option.

### 2.2. The new library

In order to face the limitations of the Matisse profiling tool, we developed a new DDT library, which is constructed in a modular and object oriented way. This subsection covers the main features of the new library.

In today's object oriented application designs, a modular way of building the software is adopted and this also includes the application's data handling. The latter, needs a place to store its data and some operations to access them. This implies the use of an abstraction layer between the application and its data. The aforementioned abstraction layer is called abstract data type (ADT) and it consists of the methods that can be used to handle the stored data. The application does not need to be aware of the underlying implementation of these methods. It only cares about the interface and the methods that it can call to serve data storage and access re-

### Table 1

<table>
<thead>
<tr>
<th>Feature</th>
<th>Matisse profiling tool</th>
<th>New DDT library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract data type (ADT)</td>
<td>Not supported</td>
<td>Supported</td>
</tr>
<tr>
<td>DDT implementations provided</td>
<td>Limited (Unsorted Lists only)</td>
<td>Extended (Sorted Lists, Trees, Sets, and more)</td>
</tr>
<tr>
<td>Extension of the DDT library</td>
<td>Not easy</td>
<td>Easy (due to the object oriented design of the library and the ADT support)</td>
</tr>
<tr>
<td>Combination of DDTs in more complex ones</td>
<td>Supported</td>
<td>Supported, but easier (due to the object oriented design of the library)</td>
</tr>
<tr>
<td>Integration in complex applications</td>
<td>Not easy (each object is handled as an array of basic data types)</td>
<td>Easy (the actual object is inserted in the DDT)</td>
</tr>
<tr>
<td>Interface</td>
<td>No STL compliant</td>
<td>STL compliant</td>
</tr>
<tr>
<td>Selection of specific DDT combinations to be evaluated</td>
<td>Not Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>GUI</td>
<td>Not provided</td>
<td>Provides a GUI for simple use</td>
</tr>
<tr>
<td>Benchmarks evaluated</td>
<td>NetBench</td>
<td>NetBench, MiBench, ALPBench, etc.</td>
</tr>
<tr>
<td>Benchmarks code size</td>
<td>A few KB</td>
<td>Tens of MB</td>
</tr>
<tr>
<td>Application domains</td>
<td>Network applications only</td>
<td>Modern network, multimedia applications, games, etc.</td>
</tr>
</tbody>
</table>

Please cite this article in press as: L. Papadopoulos et al., Exploration methodology of dynamic data structures in multimedia and ..., J. Syst. Architect. (2008), doi:10.1016/j.sysarc.2008.04.009
monolithic design. Fig. 2 shows that behind the interface there is a homogenous collection of elements, with a linear relationship between elements. This means that each element in the list except the first one has a unique predecessor, and each element except the last one has a unique successor. An unsorted list has its elements placed in no particular order.

Sorted Lists
A sorted list has its elements sorted in a variety of ways. For instance, a list of numbers can be sorted by value and a list of strings can be sorted alphabetically where all elements contain pointers to the actual data. By taking advantage of object oriented design techniques, all DDT implementations combining various elements from the library can be easily combined to more complex data structures. All abstract operations render the process of changing the data structure very easy, since no change in source code is needed.

Therefore, several implementations of these ADTs can be realized, each serving different design needs. These concrete instances of the abstract layers between the application and its data are referred to as dynamic data types (DDTs). For example, a stack can be implemented as an array, for high performance and dense memory layout, or as a linked list, for low memory footprint. Table 2 presents the ADTs that the library currently supports.

From the collection of supported ADTs of Table 2, the ones used in the current work are presented in Table 3. The DDTs supported by Matisse are the Unsor ted Lists, while the new library provides a plethora of new DDTs, as described in Table 3. The ADT design approach leads to a modular and object oriented design of the DDT library. The functionality provided by the ADTs is the interface of the DDTs implementation. By reusing the functionality of the ADTs, new DDTs can be constructed. Thus, the library can be easily expanded. For example, ADT Queue provides abstract functions such as enqueue and dequeue. Every DDT (e.g. Queue As Array and Queue As LinkedList) implements these functions in a different way. Therefore, the library can be expanded in an object oriented pattern, by creating new DDTs that implement these functions.

Furthermore, the DDTs of the new library are implemented in a way to be easily combined to more complex data structures. All abstract data types depicted in Fig. 1 can be used to form a single level DDT implementation combining various elements from the pool of components. For example we could combine a Single Linked List (SLL) element, a Roving Pointer and a Pointer Object component with a List ADT. The produced DDT would be a singly linked list, integrating an optimization for sequential access (roving pointer), where all elements contain pointers to the actual data. By taking advantage of object oriented design techniques, all DDT implementations can be reused to construct multilevel data structures. For instance, we could have a list of arrays of lists, or an array of lists of arrays, etc. The number of levels we can add to the DDT implementations is not limited by the design of the library, rather than the applicability and usefulness of such complex implementations. In this work we use two-level implementations only, for simplicity and effectiveness, which are shown in Table 3.

In contrast to the design of our library, Matisse uses a more monolithic design. Fig. 2 shows that behind the interface there are 10 implementations that are not composable. Furthermore, Matisse handles application's objects as arrays of basic data types. This means that every class of the application must be decomposed into arrays of some basic type like integer. That is the form that Matisse stored the applications objects and because of that it cannot be used easily in modern real-life applications. On the contrary, in our library the actual object of the application is inserted in the DDT. This is a more flexible way to treat objects of complex applications, comparing with Matisse.

Moreover, a GUI environment is developed, as a user interface that allows the convenient handling of the DDT exploration process. From a file of specific format our tool reads the DDTs that are present in the application, and then displays these DDTs along with the available alternative implementations for each one, as depicted in Figs. 3 and 4. In this example there are two data types, a List and a Queue with 10 and 2 possible implementations accordingly.

Using the controls the designer can choose to either test all possible combinations between the implementations of the two DDTs, or specific implementations can be selected for both DDTs. The GUI informs the designer of the current status in the DDT exploration process and also displays compilation and runtime information. The output is a file containing all Pareto points that characterize

| Table 2 |
|———GLfloat———|———GLfloat———|
| ADT | Brief description |
|———GLfloat———|———GLfloat———|
| Stack | A stack can be visualized as a pile of objects (data). Objects are added to and removed from the top of the pile |
| Queue | A queue is a pile in which items are added on one end and removed from the other |
| Deque | A deque extends the notion of a queue. Here, items can be added or removed from either end of the queue |
| Unsor ted Lists | From a theoretical point of view, a list is a homogenous collection of elements, with a linear relationship between elements. This means that each element in the list except the first one has a unique predecessor, and each element except the last one has a unique successor. An unsorted list has its elements placed in no particular order |
| Sorted Lists | A sorted list has its elements sorted in a variety of ways. For instance, a list of numbers can be sorted by value and a list of strings can be sorted alphabetically |
| Sets | An unordered collection of distinct values (items or components), chosen from the possible values of a single data type, called the component (base) type |
| Multisets | A multiset is a set in which an item may appear more than once |
| Tree | Non-linear information structures that are often used to represent a hierarchy. Many kind of trees are supported like Binary Search Trees, N-ary Trees, AVL Trees, Search Trees, M-Way Search Trees |

| Table 3 | Used abstract data types and their implementation variations |
|———GLfloat———|———GLfloat———|
| ADT | Data structures (ADT implementation variations) |
|———GLfloat———|———GLfloat———|
| Stack | Stack As LinkedList |
| Queue | Queue As LinkedList |
| Deque | Deque As LinkedList |
| Unsor ted List | List As Array |
| SortedList | List As Array Embedded |
| DLL LinkedList | DLL LinkedList |
| SLL LinkedList | SLL LinkedList with Roving Pointer |
| DLL LinkedList with Roving Pointer | DLL LinkedList with Arrays |
| SLL LinkedList with Strings | SLL LinkedList with Arrays and Roving Pointer |
| DLL LinkedList with Arrays and Roving Pointer | DLL LinkedList with Arrays and Roving Pointer |
| SortedList | List As Array |
| DLL LinkedList | DLL LinkedList |
| DLL LinkedList with Roving Pointer | DLL LinkedList with Arrays |
| DLL LinkedList with Strings | DLL LinkedList with Strings and Roving Pointer |
| DLL LinkedList with Arrays and Roving Pointer | DLL LinkedList with Arrays and Roving Pointer |

Please cite this article in press as: L. Papadopoulos et al., Exploration methodology of dynamic data structures in multimedia and ..., J. Syst. Architect. (2008), doi:10.1016/j.sysarc.2008.04.009
the application’s behavior along with a separate file only for Pareto optimal points. Furthermore, our tool [13] records the memory required by each DDT and also all accesses to memory are logged and categorized in reads and writes. This makes it easy to calculate the energy consumption as explained in Section 4.

The aforementioned features allow the new library to be used in many different application domains, such as network, multimedia and video games. Also, dynamic applications from several benchmarks have been chosen, such as NetBench [11] and MiBench [12] and refined with the DDT exploration methodology.

3. Methodology for optimal DDT exploration

In this section, the automatic exploration methodology for selecting the best-for-each-case data structures is presented and explained. Fig. 5 shows the flow of the methodology used for the optimal data structure exploration. The methodology presented here is more general than the previous one presented in [3], which refers specifically to the network application domain. The basic principles of the methodology described in this work and the ones
The common framework of the library of data structures must be plugged into the source code under exploration. This means that the operations regarding the dynamic data storage and retrieval must be replaced by the ones used by the library. If the application uses the STL interface (Standard Template Library [10]), then this step should be relatively easy, as the library is STL compliant. This is the case presented in Fig. 6. On the other hand, in case the application does not support the standard STL interface, the source code needs to be modified in order to support the library’s interface.

The fact that the interface of the library can be easily plugged in applications that use STL is very important. In such a case, the exploration of applications of huge code size can be easily performed without changing the source code of the original application. Thus, the exploration is fully automated. On the contrary, complex applications were practically impossible to be optimized with the previous methodology (i.e., DDT exploration methodology with Matisse). Additionally, since the new library supports many more DDT implementations, as shown in Tables 2 and 3, the presented methodology can be applied to a wide range of application domains.

The second step is the identification of the dominant DDTs of the application. An application may contain many data types, for example it can have two lists and several stacks. All these DDTs should be implemented using the library. Exploring every possible combination of implementations for each one of the DDT can be unmanageable in the case of a large number of DDTs. For example, consider a code that utilizes five unsorted lists. The UnsortedList ADT has nine different implementation variations in the library (SLL_ListAsLinked, DLL_ListAsLinkedList, ListAsArray, SLL_ListAsLinkedList with Roving Pointer... – as mentioned in Table 3), thus the possible combinations for exploration are $9^5$, a number large enough to delay the design process. An alternative approach, used in the current work, recognizes the dominant DDTs, meaning those that play the most significant role in the application, and then explores the combinations of these ones only. The total number of accesses for each individual data type and the number of data objects hosted by each one can be used to identify the dominant DDTs.

The last step is the DDT exploration and is fully automated. The GUI environment of the library allows the convenient handling of the DDT exploration process. The user can select the DDT combinations to be evaluated (some specific or all possible) and the library automatically evaluates these combinations. The fact that specific combinations can be selected to be evaluated is a useful feature for reducing the exploration time. However, in case the exploration is not exhausted, the application may not be fully optimized.

The fact that the new library is based on ADTs, allows the designer to automatically distinguish the DDTs that are compatible with application’s data structure. For example, if the data structure of the application is Queue, only DDTs that correspond to the specific DDT will be evaluated. This feature was not supported by the previous methodology, where all the available DDT implementations of the library were evaluated in any case. In such cases, the exploration time is reduced comparing with the previous one. Exploration time is a critical factor when applications that contain a large number of DDTs are optimized.

The output of this step is a file that contains information about the DDT combinations tested, as explained in the previous section. Specifically, the number of accesses, the memory footprint, the performance and the energy consumption are used to evaluate each combination.

### 3.2. Pareto optimal exploration

At the second stage of the presented methodology, the Pareto charts are implemented, according to the design constraints. Every
point of the Pareto chart corresponds to a different DDT combination.

This step is fully automated. The output file of the previous stage of the methodology is used as an input to a script that produces the Pareto charts and calculates the Pareto optimal points. Thus, the designer can make trade-offs between the memory number of accesses, the memory footprint, the performance, and the energy consumption, by selecting different points of the Pareto optimal curve.

Since the new DDT library provides more DDT implementations comparing to Matisse, our exploration is more thorough. Using the presented methodology, the designer can obtain Pareto optimal solutions that were impossible to reach with the previous methodology, as can be observed in the experimental results.

4. Experimental results

Employing the new hierarchical and modular library, we performed DDT exploration in a number of dynamic applications from the multimedia and network domains. Thus, we evaluated the effectiveness of the new library under six realistic case studies.

In this series of experiments, we consider a memory hierarchy consisting of an external MICRON [18] SDRAM memory operating at 256 MHz that has a size of 8 MB. That said, we used the read/write memory access info provided by our tool chain to feed MICRON models and calculate energy consumption. For the execution time we used call to the system time() function calculating the total CPU time for each simulation. The performance results were obtained on a Pentium 4 3.2 GHz with 1024 MB RAM. All the performance results presented here are average values after a set of 10 simulations for each application.

4.1. DDT exploration results

We performed DDT exploration, in six benchmarks selected from multimedia and network application domains: (i) a 2D Racing Game [14], (ii) Astar Algorithm [15], (iii) a tile game called Comboling [16] and (iv) a 3D environment builder called Simblob [17]. The benchmarks selected from the network application domain are: (v) Dijkstra Algorithm, taken from MiBench suite [12] and (vi) Weighted Fair Queuing (WFQ) algorithm, a modified version of Deficit Round Robin (DRR) application taken from NetBench [11].

The Pareto points, which are automatically extracted as the output of the DDT exploration methodology illustrate the trade-offs of memory accesses vs. memory footprint and performance vs. energy consumption. These factors are usually contradictory, thus Pareto optimal points can be usually obtained.

4.1.1. DDT exploration applied to a 2D Racing Game

The first multimedia application is a Racing Game, which contains a number of objects moving on a field according to a car’s movement. These objects are stored in an Unsorted List data structure and are accessed in a complex pattern in order to change their position, according to the car’s specific coordinates on the field. We can notice from Table 2 that the ADT of this data structure is the Unsorted List and as shown in Table 3, there are 10 different DDT implementations of the Unsorted Lists. We explored all these DDTs using the new DDT library.

The Pareto optimal points of memory accesses vs. memory footprint are displayed in Fig. 7. It can be seen that the Array with Pointers (AR(P)) is the DDT with the smallest memory footprint, while the Single Linked List (SLL) is the one with the smallest number of memory accesses. Thus, the designer can make a memory accesses vs. memory footprint trade-off between the two Pareto optimal points. For performance vs. energy consumption there is one Pareo-point, as can be seen in Fig. 8. The Double Link List (DLL) leads to less execution time and energy consumption. A comparison with the original DDT implementation, of the 2D Racing Game, which is the Single Linked List, shows that by selecting the Double Linked List, we achieved 5.8% less energy consumption, while Array with Pointers implementation leads to 44% less memory footprint.

4.1.2. DDT exploration applied to the Astar Algorithm

Astar is a graph search algorithm which employs a heuristic estimation of the distance from a source to a destination node. The algorithm is used for path finding in several video games. It contains three dominant DDTs: The first one is the m_Successor, which is used to store the nodes that have not yet been expanded for possible finding of an optimal path. The second one is the m_OpenList. This is a sorted list that is used for storing the nodes according to a specific criterion. The last one is the m_Closed, in which the nodes that have been expanded are stored. For the m_Closed and the m_Successor DDTs the corresponding ADT is the Unsorted List, for which 10 implementations are provided by the library. For the m_OpenList the corresponding ADT is the Sorted List and we evaluated 10 DDT implementations.

Figs. 9 and 10 depict the Pareto points for the two optimization objectives. For memory accesses vs. memory footprint, there is one Pareto optimal solution. In respect with the performance vs. energy consumption, trade-offs can be implemented between different Pareto optimal points. The SLL, SLL SLL combination is the one that leads to optimal memory footprint and less energy dissipation. However, by using AR(P), AR(P), S(AR) we experience a performance improvement by 9.6%.

4.1.3. DDT exploration applied to Comboling

We performed the DDT exploration methodology to a 2D tile game, called Comboling. The game contains a grid of tiles, which
are stored in an unsorted list and accessed in a random pattern. It can be easily found that the Unsorted List is the dominant DDT of the application.

Fig. 11 shows the Pareto optimal points of memory accesses vs. memory footprint. Single Linked List (SLL) is the original DDT implementation of the application. We found that this is the DDT with the most accesses. Using the Array with Pointers (AR(P)), we achieved 83% less memory accesses in comparison with the original DDT implementation. The Pareto optimal point of performance vs. energy consumption is shown in Fig. 12. It can be inferred that the Array with Pointers AR(P) implementation leads to less energy consumption and optimal performance.

4.1.4. DDT exploration applied to a 3D application

We have also evaluated our approach in a 3D application named Simblob [17] that is a 3D environment builder. A screenshot of Simblob's environment is depicted in Fig. 13. The user can design a variety of natural elements like mountains, lakes, forests, etc., that have their own impact in the simulation environment. Simblob uses the OpenGL and GLUT graphics libraries to render the objects on the screen. The source code of Simblob contains a single linked list that holds the water source instances created by the user. During the execution the water sources list is traversed, implying a sequential access pattern most of the time, a fact that favors linked list implementations (see Fig. 13).

From Fig. 14 it can be deduced that there is one Pareto point which leads to less memory accesses and memory footprint. This is the Single Linked List (SLL), which is the original DDT implementation of the application. However, in respect with the performance vs. energy consumption there are three optimal Pareto points. For instance, using the Array with Pointers the performance gains are 33%. Thus, the designer can make trade-offs between the performance and the energy consumption (see Fig. 15).

4.1.5. DDT exploration applied to Dijkstra Algorithm

Dijkstra is a well known algorithm that solves the shortest path problem in a direct graph with nonnegative edge weights. The algorithm is useful for finding the most efficient routing path in intra-domain computer networks. Every time a new network node is reached, all the adjacent nodes are inserted in a queue. When the path to a node is examined, it is withdrawn from the list. The dom-

![Fig. 9. Memory accesses vs. memory footprint Pareto points the Astar Algorithm.](image)

![Fig. 10. Performance vs. energy consumption of Pareto points of the Astar Algorithm.](image)

![Fig. 11. Memory accesses vs. memory footprint Pareto points the Comboling.](image)

![Fig. 12. Performance vs. energy consumption Pareto points of the Comboling.](image)

![Fig. 13. Simblob: 3D environment builder.](image)

![Fig. 14. Memory accesses vs. memory footprint Pareto points of the Simblob.](image)
inant data structure of Dijkstra Algorithm is the class _QITEM which represents a node of the graph stored in the queue. The DDTs implementations we evaluate are the ones that correspond to the ADTs Queue and Stack.

The original DDT implementation of the application is Queue as Linked List. Fig. 16 shows that there are three Pareto optimal points, which enable the trade-offs between memory accesses and memory footprint. In respect with the performance and the energy consumption, there is only one Pareto optimal point, i.e. the Queue as Array (Q(AR)) which leads to less energy dissipation and execution time, as well (see Fig. 17).

4.1.6. DDT exploration applied to WFQ algorithm

Weighted Fair Queuing (WFQ) is a common packet scheduling algorithm which lets a number of flows share the same link. It is implemented in various switching devices (e.g. CISCO 870 series). This algorithm is the most popular approximation of generalized processor sharing (GPS) scheme, in which every flow with a non-empty queue, at any given time, is served simultaneously and the bandwidth is equally distributed to each flow. The dominant data structures in WFQ are two: The class _Packets_, which is used to encapsulate the information of the packets to be scheduled in queues. The corresponding ADT of this class is the Queue and _Unsorted List_. 10 DDT implementations are explored.

Figs. 18 and 19 present the Pareto optimal points for the WFQ. The Pareto optimal combinations for the memory accesses vs. memory footprint are displayed in Fig. 18. The first DDT of each Pareto point (i.e. the _SLL_) is referred to the class _Nodes_ and the second (i.e. Q(AR) and Q(SLL)) is referred to the class _Packets_. The same applies to the Pareto optimal points of Fig. 19. We can notice that trade-offs can be applied for both optimization factors. For instance, by selecting (AR(P), Q(AR)) performance is increased by 22% in comparison with the original implementation. (SLL, Q(SLL)) is optimal in terms of energy consumption.

4.2. New library evaluation

In this subsection, we evaluate the effectiveness of the library, based on the DDT exploration we performed in the experimental results. We point out the critical differences between our approach and the one used in Matisse, from the case studies perspective.

In the Racing Game, we took advantage of the STL compatibility feature of the library. Plugging the library interface in the source code of the Racing Game was automatic, due to the fact that the application uses STL. Therefore, we avoided code modifications that would have been necessary if we had used Matisse profiling tool.

In Dijkstra application and in WFQ, we evaluated DDTs that correspond to the Queue. Thus, we explored only the DDTs that correspond to the functionality of a queue and not irrelevant DDTs. However, if Matisse had been used, many more DDTs would have been evaluated. Thus, with the methodology we present, the exploration space is reduced.

We evaluated DDT implementations that are not supported by Matisse, such as Queue, Stack and Sorted Lists, which were used in Dijkstra and WFQ, respectively. The Pareto optimal points obtained...
in Dijkstra, could not have been obtained with the previous methodology. Additionally, the exploration of Astar is not possible with the previous methodology, since Matisse did not support the sorted lists. Therefore, the new methodology extends the range of applications, in which DDT exploration can be applied.

Finally, we applied our methodology in a large number of applications of various complexities and from different application domains and from different benchmarks. The previous methodology was not evaluated using such an extensive exploration.

5. Conclusion

A systematic DDT exploration methodology based on a novel library for deriving Pareto optimal curves is presented. We applied the improved exploration methodology to six benchmarks from the multimedia and network application domains. The new methodology allows the optimization of a wide range of dynamic applications, by performing trade-offs using a set of Pareto optimal curves.

Acknowledgement

We would like to thank Christophe Poucet (IMEC, NES group, Leuven, Belgium) for his help and support in the development of profiling framework.

References


Lazaros Papadopoulos received his Diploma in Electrical and Computer Engineering from the Democritus University of Thrace, Greece, in 2005. He is currently a post graduate student in the VLSI Design and Testing Center in the Democritus University of Thrace. His research interests include Networks-on-Chips and system-level design optimizations in embedded platforms.

Christos Baloukas received his Diploma and M.Sc. Degree in Electrical and Computer Engineering from the Democritus University of Thrace, Greece, in 2004 and 2006, respectively. He is currently a Ph.D. researcher in the VLSI Design and Testing Center in the Democritus University of Thrace. His research interests include dynamic access and storage optimization on communications applications for low power and high performance, embedded systems and high-level design optimizations.

Dimitrios Soudris received his Diploma in Electrical Engineering from the University of Patras, Greece, in 1987. He received the Ph.D. Degree in Electrical Engineering, from the University of Patras in 1992. He is currently working as Assoc. Professor in Department of Electrical and Computer Engineering, Democritus University of Thrace, Greece. His research interests include low power design, parallel architectures, embedded systems design, and VLSI signal processing. He has published more than 170 papers in international journals and conferences. He was leader and principal investigator in numerous research projects funded from the Greek Government and Industry as well as the European Commission (ESPRIT II–III–IV and 5th, 6th and 7th IST). He has served as General Chair and Program Chair for the International Workshop on Power and Timing Modelling, Optimisation, and Simulation (PATMOS) and General Chair IFIP/EURO VLSI-SOC 2008. He received an award from INTEL and IBM for the project results of LPCD #23256 (ESPRIT IV). He is a member of the IEEE, the VLSI Systems and Applications Technical Committee of IEEE CAS and the ACM.

Please cite this article in press as: L. Papadopoulos et al., Exploration methodology of dynamic data structures in multimedia and ..., J. Syst. Archit. (2008), doi:10.1016/j.sysarc.2008.04.009