

Energy Efficient Middleware: Design and Development for Mobile Applications

Fatimah Abdualaziz Almusalli

King Faisal University, KSA

215002782@student.kfu.edu.sa

Noor Zaman

King Faisal University, KSA

nzman@kfu.edu.sa

Raihan Rasool

Victoria University Melbourne

Raihan.Rasool@live.vu.edu.au

Abstract— Over the recent years, the popularity of smartphones has increased dramatically. The advanced integrated technology in smartphones like GPS, high-speed CPU, a real world coloured display, Wi-Fi and Bluetooth etc. All these within small size light weight device attracts people a lot to obtain them. The stated capacities of those components motivate developers to create millions of useful applications. However, smartphone devices are energy constraint as they rely on limited battery power supply that has not been increased at the same pace to support the power demands. As both the hardware and the software tend to drain the battery power, the demand for energy efficient applications has increased to keep the mobile devices useful. Optimization related to memory data access create significant difference to performance and power consumption of broad range of data-intensive application. Memory Data layout transformation represents a very interesting class of optimizations. Transform Array of Structure (AOS) to Structure of Array (SOA) is one of the commonly applied and recognized transformation. The transformation reduces the memory access count and subsequently reduces the memory access energy. Thereby, we introduce data layout transformation service as solution to minimize the power consumed by application. The Service will convert the data layout in memory from AOS to SOA. The conversion will reduce the power consumed by memory and processor. Eventually, result in efficient and extended battery life.

Keywords— Data Layout Transformation DLT, Mobile Computing, Energy Efficiency, Array of structure AOS, Structure of arrays SOA

I. INTRODUCTION

Despite the market predominance of smartphones and exponential growth of their applications market, their utility remains limited by their battery life. The increase in power cost and the need for more power efficient applications to keep the smartphone useful required the developer to add more efforts during the applications development and maintenance.

Today smartphones are built using most advanced technologies, Full HD (High Definition) and Quarter High Definition display (QHD), powerful processors such as Snapdragon 820 and Exynos 8890 that bring with it more power efficiency, speed, connectivity and image processing. Camera resolutions up to 23MPs (Megapixel) and fingerprint scanner [1]. However, despite those a remarkable pro of the smartphones, one drawback is that they have a limited battery

life. Integrating powerful features and more technology in smartphones demands more power consumption. Comparing the battery size for the most popular 2015 smartphones, we find that median battery size for smartphone is 3000 mAh as in Galaxy Note 5 and LG4. The maximum battery size is 3760 mAh as in Moto X Force compare with 2014 Nokia Lumina 1520 has 3400 mAh [2]. We see that battery size increase slowly. To overcome the shortage of the battery life, scientists conduct many researches to get more power efficient hardware component, such as smart display material that slashes the energy required to power the smartphone's screen, where the 90% of the battery power consumed to illuminate its display [3]. Moreover, an aluminum battery that could be charged in less than one minute and keep going for more than seven times as long as lithium-ion battery [4]. Although the advanced technology is trying to alleviate smartphone battery limitation, it couldn't prevent the applications from unproductively consuming the battery power. Therefore, improving developers' ability to engineer more energy efficiency application is essential to reduce the power consumption.

Smartphone memory represents a critical performance bottleneck. Many processing and input/output units mostly communicate over the main memory. The Processor and the memory consume a large part of battery power almost 15% in the overall system and this percent increases with the usage. SD card and RAM consume around 7-10% of the total battery power during performing read and write memory operation. Memory and CPU are core parts of the system and required to be awake as long as the smartphone is switched on [5]. During development of the smartphone, the gap between the memory and the CPU frequency has increased in the last decades. To overcome this gap multilevel of memory hierarchy introduced to enhance the performance [6]. However, this improvement is useful only when memory efficiently utilize. The poor usage of memory waste the memory bandwidth and increases the memory access time. Further, it increasing the power consumption by memory as well as the processor. An appropriate data structure layout can optimize the performance of a memory by improving access pattern. Transform how the data layout in memory is the most effective optimization strategy. The most commonly applied and recognized transformation is the Array of Structure (AOS) to Structure of

Array conversion (SOA) [7]. This motivates us to introduce Data layout transformation service. An automatic service that transforms the data structure in the memory to improve the memory performance and save the battery power. The rest of the paper is organized as following. In section II. We introduce the background of the project. Section III gives a summary about related work. Section IV presents the approach and implementation of proposed technique. Finally, results and discussions are presented in section V.

II. BACKGROUND

In this section, we start by discussing the issue of the power consumption by smartphone. Then we provide an explanation to Structure of array SOA and Array of structure AOS.

A. The Power Consumptions in Smartphone

The increase in the power cost raised the demand for more power efficacy in both hardware and software component. Developing a power efficient application is an important goal for a software developer as power saving effects directly the usability of the device. Fig.1 shows that 47.9% of battery drains cause by applications [8].

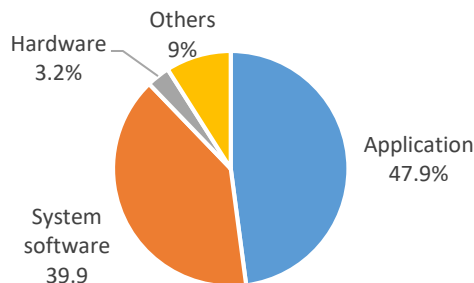


Figure 1. Distribution of battery drain by components [8].

The increasing in the power cost and power consumptions by applications motivates the researcher to conduct many different research studies on mechanisms for reducing power consumptions [9]. In [10], they developed an annotation language and middleware for energy-efficient. Their middleware, (APE) Annotated Programming for Energy-efficiency, built in two main keys. First, delay the execution of the operation until the device enters the statue that minimizes the cost of that operation. Second, describing the desired device status using an abstract model based on timed automata. Their experimental result on 63.7% on battery power saving. However, this approach did not address the trade-off between timeliness and power saving which efface the user experience. More precisely, they provided a mechanism to specify the max delay but no mechanism to determine the minimum delay. Therefore in [11] they introduce Tempus, a new paradigm to write power management policies. Many of scheduled task in the smartphone are set using an alarm. Also, many of applications used wake-up alarm either to communicate with the server, check for the application version or update the date. However, the frequent non-time critical wake-up wastes the smartphone energy. In a like manner, [12] also use delay for non-time critical system wake-ups to reduce the power

consumption. The power saved by proposed method between 2.6% and 12.5%. In [13], they study the effect of some of the common programming practice in battery life, such as network packet size, memory usage, array length access, invocations type and direct field access. They find that many of these practices can reduce the power consumptions by 10% to 30%. In [14] they introduced dynamic flexible resolution scaling system to reduce the power consumed by the display. Their system uses ultrasonic-based approach to detect the user display distance at lower cost and with automatically decide the appropriate scaling. The energy per frame can achieve 60.5% reduction at most and 30.1% on the average when the resolution is halved. CPU and Graphics Processing Unit (GPU) usage have very significant impact on battery life, [15, 16] use Dynamic Voltage and Frequency Scaling (DVFS) with User Driven Frequency Scaling (UDFS) to reduce the power consumptions in both CPU and GPU system. Their experimental result in reducing the power consumptions of CPU by 25% compared to default DVFS and 3% compared to existing UDFS and their approach work in idle and running state of android applications. Wi-Fi needs high bandwidth for work; this requires high battery power. One of the techniques used to increase the power efficiency of wireless adapter called waked. the power efficacy estimated up to 115% when applying this technique [17]. in [18] they work on optimizing the web browser to reduce the power consumptions. They optimization focused on reduces the web page loading time, predict the user reading time and adoptee the wireless adapter status. Their experiment shows 30% power consumption reduction during the web browsing. In [19] they focused on study and analysis the power consumption while shooting and streaming video in real-time. They try to optimize the video to reduce power consumption by bundling the video size. As a result, bundling reduces the wireless communication power by 80% but the net effect limit to 20% because of the role of the other hardware in the total power draw. HTTP request one of the most energy consumption operation related to network, almost 80% of network energy consumption related to HTTP. In [20] they propose bundling to optimize (HTTP) Hypertext Transfer Protocol request. Their approach achieves 15% on the power saving at the application level. In [44], they study the relationship between the GPS signal strength and the power consumed by the localization applications. Their experiment shows that under a good signal strength 13% of battery is drained while 38% drained under weak signal strength. The result demonstrates that with low GPS signal more battery power drain compared with strength signal consume less battery power. In [45], they introduced an alternative solution to track the user location. They use the combinations of other sensors available in the smartphone to provide the service. The magnetometer, the gyroscope and the accelerometer have been used together to provide the locations of the moving object. The proposed method achieved 20% battery saving. Moreover, using GPS fully battery drain in 9.09 hours while using their applications drains the battery completely in 13.63 hours. The self-adaptive technique that reacts depending on battery level, status and context information at the real time discussed in [21].

They introduced a framework that uses analyzer engine which decides according to battery level and context information which appropriate profile to activate to save battery power. Using this framework, the power consumption reduced to 40%. Fig.2 shows the major components that utilize the smartphone battery power [24]. Table.1 summarize the different techniques introduced in this section and present the amount of the power saving in percent.

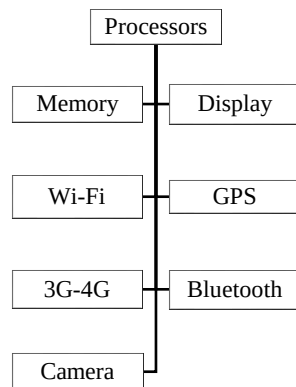


Figure 2. Smartphone Power Consuming Components [22]

B. Array-of-structure (AOS) VS Structure-of-Array (SOA)

Data layout can be defined as the form in which the data could be organized and accessed in the memory when the data have multi-value as in the 3D point. The appropriate data layout has a crucial issue in the development intensive computing application or graphics processing unit application. The performance of the application may drastically differ because of the use of different type of data layout. For data layout, there are two main choices. The first one is the Array of Structure. The Second one is the Structure of Array. The AOS data layout Fig 3b. store data fields in continuously in memory while SOA data layout Fig.3a stores fields using separate area. Fig.4a shows the syntax for AOS and Fig4.b shows the syntax for SOA. The SOA have better locality in the case all fields of the structure do not require by the application in the same time frame. Alternatively, the AOS has used if the application required all the field of the structure at one time. [25]. Due to the different layout, the memory access during the computation are different significantly [26]. AOS make the code more readable and easy to maintain but it is not the efficient layout for the processor. SOA Provide efficient data access but code become hard to maintain and it against the intuition of understanding the object structure. The issue includes the trade-off between the maintainability and readability of the code and between the performances of the device [27]. Because the access syntax is differing for the two layouts, switching between them is labor intensive. If each data moved to the cash and accessed multiple times, then data will stay in the cache and reused many times that will decrease the number of cache misses. Because SOA stores each filed in a different array, it provided the shortest runtime and better locality of the references. The SOA provides better usage to the memory cache. This result in less cache misses.

TABLE 1. BATTERY POWER SAVING TECHNIQUES CRITICAL SUMMARY.

Reference	Target	Optimization Method	Power Saving
[22]	Activation state of the smartphone system	Appropriate placement of wake lock acquires and release functions	32%
[10]	Most hungry power component	Delay the execution and timed automata	63.7%
[12]	Alarm-induced Wake-ups	Delay for non-time critical system wake-ups	2.6% - 12.5%.
[13]	Memory	Avoiding referencing the array length in the loop	10%
[13]	Memory	Use direct field	30% - 35%
[13]	Memory	Use static invocations	15%
[14]	Display	Dynamic flexible resolution scaling system with ultrasonic chips.	30% - 60%
[15]	CPU	Dynamic Voltage and Frequency Scaling with User Driven Frequency Scaling	25%,3%
[16]	GPU	Dynamic Voltage and Frequency Scaling with User Driven Frequency Scaling	Battery lasts twice the time
[17]	Wi-Fi	Waked: reducing the idle power	115%
[18]	Web browsing	Reorganize the computation sequence and predict the user reading time of web pages	30%
[19]	Video Application	Bundling the video size.	20%
[23]	GPS	Use the combinations of other sensors	20%
[21]	Applications	Control of hardware and software depending on battery level	40%.
[20]	HTTP requests	Automatically detecting and bundling multiple HTTP requests.	15%

In contract, the AOS it does not provide good performance if particular fields not used all at once in the same time.



Figure 3. The Array of Structure(b) vs. the Structure of Array layout(a).

```
struct S {float x;      struct S {float x[N];
        float y;        float y[N];
        float z;        float z[N];
    };                  };
struct S myData[N];     struct S myData;
```

(a) (b)

Figure 4. The Syntax for Define the AOS (a) and the SOA (b)

III.RELATEDWORKS

Optimizing memory has become increasingly important to exploit the full computational power of modern GPU and maximize the reuse of data in memory. In [13], they investigate the impact of memory usage and the power consumption. They found that high memory usage consumes more power also may increase the power consumption of the application. Moreover, the insufficient memory may increase the power consumption by another component in the smartphone such as Wi-Fi and 3G. Figs.5 shows the energy consumption per access of array cell. As graph illustrate, the average power consumption per each array cell access increase when memory usage increase. The power consumption per each array access increases 21.7%.

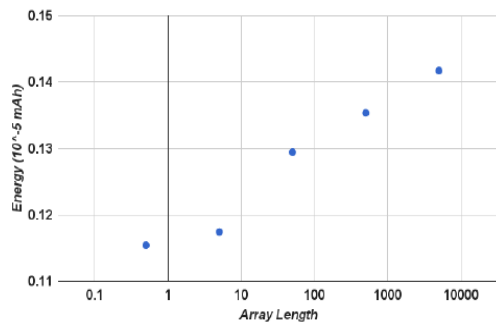


Figure 5. Power Consumption at Different levels of Memory Usage [13]

Modern DRAM designed to transfer bursts of large rows or lines of date. The Poor usage of bursts results in wasted memory bandwidth. Transform the data layout in memory is the most effective optimization strategy. The CPU and memory are the main part of the system. All other component and application are depending on them to process and store the data. The power consumed by the both part can increase the power consumption by the application and other sensors if they utilized inefficiently. The most commonly applied and recognized transformation is the Array of Structure to the structure of array conversion. Testing the conversion in multi-threaded system result in 5% to 30% speedup in different benchmark [7, 28]. If the data are stored using AOS layout, this will require data to be rearranged before performing the operation. If the data was stored in the SOA, loading data to the

register will be easy. The Data layout transformation can be used to optimize the access time of memory. This lead to reduce the power consumption of the memory. Data layout transformation introduced in the heterogeneous system, embedded system etc. as a solution to optimize the performance. In [29] they introduce a method to improve the performance of geometry computations by transposing 3D data vector. The transformation is done using shuffles. In [30], they present an optimization infrastructure that automatically determines an improved data layout for OpencCL program written in AOS layout. The framework consists of two algorithms. The first algorithm establishes a graph-based model. The model used to split the AOS input into multiple several clusters of fields. The second algorithm selects a good data layout per cluster such as AOS, SOA or an intermediate layout using a decision tree. In [27] the another proposed a framework which enables automatic data layout transformation. The framework is transparent to the programmers. The experiment with framework results in up to 177% improvement in the performance. In [31] the author present data layout transformation approach. The source code converted to the intermediate language then analysis to identify the vectorizable structure that appropriates for the conversion. One drawback of the proposed approach that the cost may be larger than the benefit and need to be optimized. In [32] they introduce Xevolver that used for optimizing the performance of (HPC) the high-performance computing application by transform the data layout. The Xevolver work by enabling the user to define a custom rule for transformation. This rule applied to an existing code and target the specific data representation. Then the rule consistently and mechanically translated the code to the desired layout. In [33] They study the performance of using Xevolver code transformation framework by defining a rule that transfers AOS to SOA. The performance in the CPU and GPU increase. The improving in the GPU is more noticeable as it has a smaller memory than the CPU. The Transformation reduces the memory access and increases the hit ratio which results in better memory utilization. In [34] they introduce transformation method based on the binary code analysis. They work with binary code of the application that was analyzed using (MAQAO) Modular Assembly Quality Analyzer and Optimizer.

We conclude this sections by summarizing the techniques introduced prior for converting the AOS to SOA in table.2.

TABLE 2. DATA LAYOUT TRANSFORMATION TECHNIQUES SUMMERY

Contribution	Result
[13] Investigate the impact of memory usage and Power consumption.	The power consumption per each array access increases 21.7%.
[28] Test the performance of AOS to SOA in multi-threaded system	5% to 30% speedup in different benchmark
[29] Improve the performance of geometry computations by converting From AOS to SOA using shuffle	The SIMD implementation using shuffles performs better than the best serial implementation.

[30] Automatically determine an improved data layout for OpenCL program written in AOS layout	Speedup of up to 2.83 over different AOS programs
[27] Automatic data layout transformation for heterogeneous many-core Systems	Up to 177% improvement in the performance (Hardware Based)
[31] Data layout transformation approach with analysis technique to detect where the transformation can be suitable.	Improves performance up to 49.5% on average. (cost for implementation high)
[33] Study the performance of using Xevolver code transformation framework by defining a rule that transfers AOS to SOA	The Transformation reduces the memory access and increases the hit ratio results in better memory utilization
[34] Transformation method based on the binary code analysis.	Transforming and evaluating data layout transformations

IV. APPROACH

In this section, we introduce the method used to achieve the transformation of source code tool. Implementation stage consists mainly of two phases. Phase one was designing the transformation tool. The transformation tool transforms the source code from the array of structural to structure of array. The second phase was designing small android application to test the transformed code.

A. Designing the Data Layout Transformation tool

This phase consists of two parts. The first part will introduce the way to convert the array of structure to structure of array. The second part will discuss the method for designing source to source transformation tool.

To perform the transformation of the code we use Intel SDLT library (Intel® SIMD Data Layout Templates) [35]. This library provides a method for the user to write the code in the form of array of structure, but the data stored in the memory in the form of structure of array. The library provided an SOA container that encapsulates the AOS. Using this container enable the compiler effectually to generate the data in the form of structure of array. Fig.6 shows code in the form of AOS. Fig.7 shows the same code converted to SOA using SDLT library. To use Intel SDLT library, we need first to declare the SDLT library. The second change declared the structure to be primitive and identify its members. The Primitive represent the data that we want to process in the form of SOA. The next step was using the SDLT container. The container encapsulates the array of the primitive and abstracts the in-memory data layout. Finally, obtain the accessors method to access the data member of the structural. The accessors provide an array interface to the subscript underlying data [36].

Design the tool that perform the source to source transformation required two-step. The first step, search and locate an interesting node corresponding to the source code. The second step performs the necessary change to that node. To implement this tow steps, we use the LLVMClang compiler. The Clang project provide powerful and friendly library for

C++ compiler also provide platform to build source level tool based on C++ language. The most interesting and helpful library was the LibTooling library. LibTooling library provides many different tools that help for refactoring the source code and do source to source translation. In our project, we mainly focused on two tools. ASTmatcher and LibTooling library tools. AST matcher library is a tool that used to locate the interesting node in the source code structure. [37]. The Libtooling library then used to make the change in the underlying source code corresponding to matched node. [38]. The next paragraph discusses in detail how we use both of this library.

The first step for designing the tool was understanding and studying the AST of the source code. AST (abstract syntax tree), is tree represent the abstract syntactic structure of the source code that written in programming language. Each node in the tree represents a construct occur in the source code. Understanding the AST help to correctly locate the interesting node from the source which in our case was the structure and its member. Fig.8 represent the AST corresponding to the code that we want to make a change it from Fig.6.

```
#include <stdio.h>
using namespace std;
#define S 500
struct RGBs {
    float red;
    float green;
    float blue;
};
int main()
{
    RGBs pixelsArray[S];
    for (int i = 0; i<S; ++i) {
        pixelsArray[i].red = i*1.5;
        pixelsArray[i].green = i*2.5;
        pixelsArray[i].blue = i*3.5;
    }
}
```

Figure 6. C++ Code Written in The Form of AOS.

```
#include <stdio.h>
#ifdef WIN32
#include<sdl_t/sdl_t.h>
#else
#include<sdl_t/sdl_t.h>
#endif
using namespace std;
#define S 1024
struct RGBs {
    float red;
    float green;
    float blue;
};
SDLT_PRIMITIVE(RGBTs, red,green,blue)

void main()
{
    sdl_t::soald_container<RGBTs> aContainer(S);
    auto pixelsArray = aContainer.access();
    for (int i = 0; i<S; i++) {
        pixelsArray[k].red() = i*1.5;
        pixelsArray[k].green() = i*2.5;
        pixelsArray[k].blue() = i*3.5;
    }
}
```

1) Define the SDLT library

2) Declare the Structure to be Primitive

3) Using the SDLT container

4) Obtain the accessors method to access the data

Figure 7. C++ Code Written in The Form of SOA.

```

CXXRecordDecl @0a03378 <line:4:3, line:8:10> referenced struct RGBs definition
CXXRecordDecl @0a03400 <col:3, col:10> implicit struct RGBs
FieldDecl @0a03400 <line:5:5, col:10> col:11 referenced red 'float'
FieldDecl @0a03400 <line:5:5, col:10> col:11 referenced green 'float'
FieldDecl @0a03390 <line:7:5, col:10> col:11 referenced blue 'float'
CXXConstructorDecl @0a03378 <line:4:10> col:10 implicit used RGBs 'void (void) throw()' inline
CompoundStmt @0a03368 <col:10>
CXXConstructorDecl @0a03378 <col:10> col:10 implicit RGBs 'void (const struct RGBs &)' inline noexcept-unevaluated @0a037c8
ParmVarDecl @0a03380 <col:10> col:10 'const struct RGBs &'
FunctionDecl @0a03350 <line:9:1, line:17:1> line:9:5 main 'int (void)'
CompoundStmt @0a03300 <line:10:1, line:17:1>
DeclStmt @0a03300 <line:11:3, col:2>
VarDecl @0a03368 <col:3, col:20> col:8 used pixelsArray 'struct RGBs [500]' callinit
CXXConstructExpr @0a03368 <col:3, col:20> 'struct RGBs [500]' 'void (void) throw()'
ForStmt @0a03360 <line:12:5, line:16:5>
DeclStmt @0a03370 <line:12:10, col:19>
VarDecl @0a03320 <col:10, col:10> col:14 used i 'int' cinit
IntegerLiteral @0a03360 <col:10> 'int: 0'

```

Figure 8. the AST for the Structure Definition.

From the AST we find that structure definition corresponds to the CXXRecordDecl. We use clang-query tool to help us to identify that node is the correct one. The Fig.9 show example of clang-query corresponds to CXXRecordDecl using Ubuntu terminal. The result from the tool highlights the code corresponding to the interesting node.

```

clang-query> match cxxRecordDecl()
Match #1:
/home/hanln/clang-llvm/llvnc/inputs/AOS.cpp:4:3: note: "root" binds here
struct RGBs {

```

Figure 9. the Clang-Query Example.

Finding the interest node "Structure" corresponding to source code was the first step. The second step was getting the name of the structure also the name of its members to use them for declaring the Primitive. Also, we use the obtained information to encapsulate the structure with the container. We use the cxxRecordDecl reference to help us to locate the rest of the information [39].

The second step was to insert the Primitive declaration include the structure name and the structure members name in the code. ASTConsumer provides an interface to read the AST produced by Clang parser and matching the node from parsed code. The Rewriter then used to write the change in the underlying source in respond to the found node. Fig.10 shows the code written to find the structure definition.

```

class StructHandler : public MatchFinder::MatchCallback {
public:
    StructHandler(Rewriter &Rewrite) : Rewrite(Rewrite) {}

    virtual void run(const MatchFinder::MatchResult &Result) {
        if(const CXXRecordDecl *d =
            Result.Nodes.getNodeAsCXXRecordDecl("CXXRecordDecl")->getDefinition()){
            if (d->isStruct()){
                std::string StructName (d->getNameAsString ());
                SourceLocation ST = d->getLocEnd().getLocWithOffset(2);
                std::string SSAAfter;
                SSAAfter = "\n"+SDLT_PRIMITIVE(" "+StructName+ " ");
                for(CXXRecordDecl::field_iterator f =
                    d->field_begin(), e = d->field_end(); f!= e; ++f){
                    std::string Mname (f->getNameAsString ());
                    SSAAfter= SSAAfter + Mname+ " ";
                }
                SSAAfter.pop_back();
                SSAAfter = SSAAfter + " ";
                Rewrite.InsertText(ST, SSAAfter, true, true);
            }
        }
    }
private:
    Rewriter &Rewrite;
};

```

Figure 10. Code for Matching Structure and Its Elements.

The next step after finding the structure definition was found the instance declaration from that structure. Looking to AST we found that corresponds to VarDecl node. Using varDecl class reference and the other function inherent from the other class such as ValueDecl. We manage to get the name of the structure object and the array defined size. After obtaining the name and array size we use the Rewriter object to add the SOA container and declare the access to the structure members. The final step to make the transformation was added the access parentheses for the structure object member. We accomplish this by using memberExpr. The final step is to convert this code into tool. To accomplish this, we use clangTool class from the Libtooling. The clangTool convert the low-level transformation logic into tool using the FrontAction. The FrontAction provide custom AST consumer and setup the Rewriter for each new file need to process. The tool need only to pass the path with the name of the source file. Fig.11 shows the tool apply to the code presented in Fig.6. Fig.12 summary the tool developing step.

```

hanlinghanlu:~/clang-llvm/llvnc$ build/matchers_rewriter Inputs/AOS.cpp --
#include <SDL2/SDL.h>
#include <stdio.h>
using namespace std;
#define S 500
struct RGBs {
    float red;
    float green;
    float blue;
};
SDL_PRIMITIVE(RGBs,red,green,blue);
int main()
{
    //RGBs pixelsArray[S];
    SDLT::soad_container<RGBs> aContainer(500);
    auto pixelsArray = aContainer.access();
    #pragma omp simd
    for (int i = 0; i<S; ++i) {
        pixelsArray[i].red() = i*1.5;
        pixelsArray[i].green() = i*2.5;
        pixelsArray[i].blue() = i*3.5;
    }
}

```

Figure 11. Using the Data Layout Transformation Tool.



Figure 12 . Step Used to Design the DLT tool.

We conclude this part by summarizing the software needed to design and develop the transformation tool. First, The Operating system. To work with LLVM compiler required one of the supporting operating systems that compatible with LLVM. [40]. We use Ubuntu 16.04.1 LTS x32bit [41]. Also, we use the terminal to type and execute the command to the LLVMClang. Also, we use LLVMClang 3.8 compiler [42].

B. Designing the Android testing Application

Java Native Interface (JNI) was used to develop the Android application that supports working with C++. JNI is a programming framework that enables java code to call and called by native application and other libraries that were written using language such as C and C++ [43]. We use Microsoft Visual studio C++ cross-platform to develop the Android mobile application with native language [44]. We made three versions of the application. One for AOS, SOA and finally one for transformed code using SDLT and we will refer to it as SOA SDLT. The three versions were used to do the test.

V. RESULTS AND DISCUSSIONS

This section presents the result we get from the implementation stage. Additionally, we introduce analysis for obtaining result.

To test and debug the Android application, Visual Studio Emulator for Android was used. The Android smartphone has API 19 KitKat, that support CPU 86x architecture with SSE (Supplemental Streaming SIMD Extensions) and AVX (Advanced Vector Extensions) instruction support. The development machine specification that used during the developing and testing stage was Lenovo Laptop with Intel core i5-3230M CPU 2.60 GHz. Installed memory RAM was 8.00 GB. And support for the 64x bit operating system.

The performance matrix that was used to evaluate the result consists of three elements. The speedup gets from processor. The information about the speedup was get using Intel compiler diagnostic report that generating after code compilation. The second element, the elapsed time that calculates time take to execute the code measure by the nanosecond. The third element, the power consumption during the execution. The power consumption measures using the power profiler mobile application.

To test the performance of the AOS and SOA and the transformed code we design a simple program that defines a structure with three elements. We use loop to ensure the access to the elements so we can measure the performance. We enforce the loop vectorization by using `#pragma simd`. This will enable us to measure the maximum performance we can get from each form of coding. Fig.13 shows the speed gain by different layouts. The SOA_SDLT get the best speedup while the AOS the smallest. the speed up of SOA_SDLT was 3x better than structure of array.

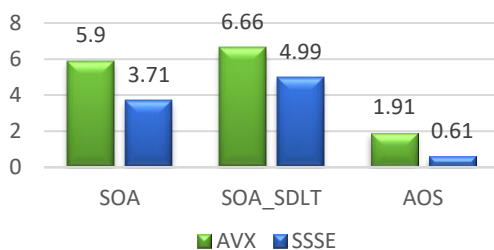


Figure 13. Speed Gain by Different data layout

We use second example [36] to test the layout of AOS that manually vectorized and SOA_SDLT to compare the speed up. The code used to normalize three points. Also, the loop was used to ensure the access of the elements of structure. The speed obtained for array of size 1000,000 for the SOA_SDLT version was 8.230 over 3.510 for manually vectorized code. The SDLT version provides 2.35x speed up over the AOS code. We use third sample application to test the elapsed time and power consumption by scaling the array size from 2000 structure element to 200,000,000. For 200 iterations and taking the average of the time. Fig.14 Shows the result obtained from the test. The smaller time is better. The Elapsed time for the SOA, in general, is better than AOS.

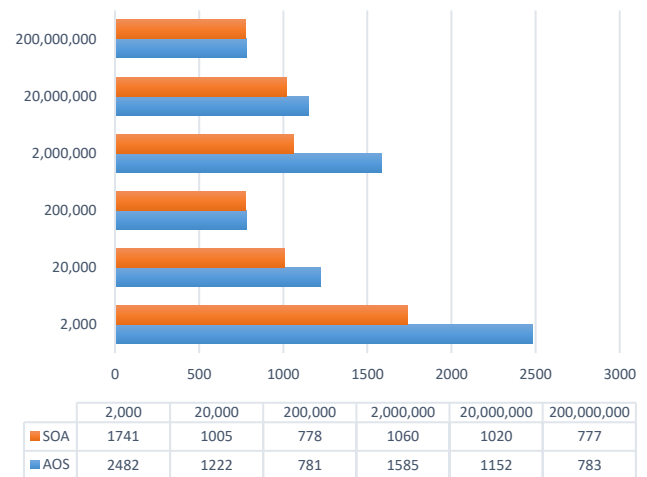


Figure 14. Elapsed Time for (a) AOS and (b) SOA

The Fig.15 show the power consumption for AOS and SOA. The graphs show the power consumption over the time. Comparing the two graphs, we can see that AOS consume more power than SOA. The test was conducted using array with 200000000 structure elements, an average of 1000 iteration for 10 min automated test using the same application used to test the elapsed time. The test was conducted using Acer Iconia A1-830.

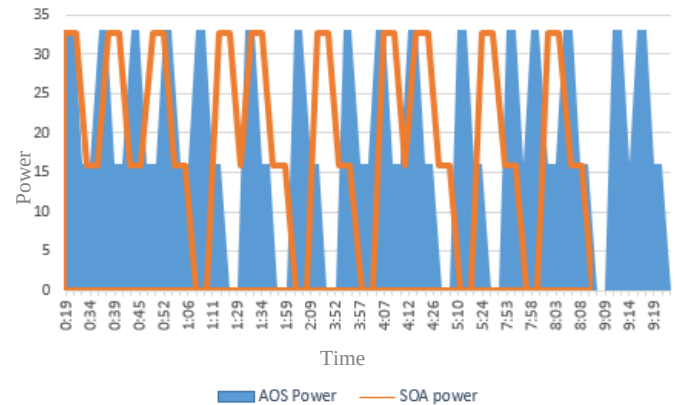


Figure 15. Power Consumed by SOA

We conclude this section by summarizing the result we get from this test. As illustrated by the test examples. The performance of the SOA is better than the AOS. The speed up we get when using the SOA layout was the double speed of AOS. Also, the DLT tool successfully detects the structure and convert the layout into SOA using SDLT library. This help the user to keep the code simple as possible and readable in the same time. The Next step is to do more test to find the effect of this speed in the smartphone battery life. Also, optimize the tool to cover more sophisticated code. This will require more study to AST and the class reference for each node in the AST. Furthermore, the performance of the code also affected by other factors. some of those factors are, the parallelism, data alignment, and CPU supported instructions. Each of those may increase or decrease the benefit gain from using SOA data layout.

VI. CONCLUSIONS

Developing power efficient applications becomes a significant goal which requires significant effort at the developer end. Handling the code segment related to managing the power-hungry resource tends to be complex and prone to error, mishandling of power hungry resource and mistakes in the code segment drains the battery power. Smartphone memory represents a critical performance bottleneck, mainly processing, input/output units communicate over the main memory. The Processor and the memory consume a large part of battery power which is almost 15% in the overall system which further increases with the usage. Data layout transformation (DLT) represents a very interesting class of optimizations. The most commonly applied and recognized transformation is the AOS to SOA conversion. Because the access syntax is differing for the two layouts, switching between them is labor intensive. To overcome stated issues, this paper presents Data Layout Transformation service. The service converts the code written in the form of AOS to source code that stores the data in form of SOA. We benefit from SOA layout by decreasing the memory access time. Eventually, this should reduce the memory power access and the CPU power consumption at the same time. This leads to minimize the power consumptions by applications and finally extend the battery life.

Based on the testing and evaluation, it was clearly noticed that the performance of SOA is better than the AOS. Also, the DLT tool designed resulting this research, successfully convert the test code to the form of SOA using SDLT. Further this research in future will improve the Data Layout Transformation tool to be applied for more sophisticated codes which may be applied for different apps such as games, image processing to further enhancing the energy efficiency and battery life of smart phone.

ACKNOWLEDGMENT

We acknowledge with thanks and appreciations to King Faisal University, Saudi Arabia for providing this precious opportunity and environment to complete this research.

REFERENCES

- [1] E. Pulcu, "Back to the future: What will a flagship from late 2016 look like?," *Android Authority*, 17-Dec-2015. [Online]. Available: <http://www.androidauthority.com/what-to-expect-flagship-phone-2016-653616/>. [Accessed: 15-Dec-2016].
- [2] G. Sims, "Which Android phones charge the fastest?," *Android Authority*, 21-Dec-2015. [Online]. Available: <http://www.androidauthority.com/which-phone-charges-the-fastest-660181/>. [Accessed: 15-Dec-2016].
- [3] R. Burn-Callander, "Is this the end of constant smartphone recharging?," *The Telegraph*, 21-Nov-2015. [Online]. Available: <http://www.telegraph.co.uk/finance/businessclub/technology/12008390/Is-this-the-end-of-constant-smartphone-recharging.html>. [Accessed: 15-Dec-2016].
- [4] S. Knapton, "Smartphones could be charged in 60 seconds with new battery," *The Telegraph*, 06-Apr-2015. [Online]. Available: <http://www.telegraph.co.uk/news/science/science-news/11518125/Smartphones-could-be-charged-in-60-seconds-with-new-battery.html>. [Accessed: 15-Dec-2016].
- [5] Naik, B. A., & Chavan, R. K. (2015). Optimization in Power Usage of Smartphones. *International Journal of Computer Applications*, 119(18)..
- [6] N. Faria, R. Silva and J. L. Sobral, "Impact of Data Structure Layout on Performance," *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, Belfast, 2013, pp. 116-120.
- [7] J. A. Stratton *et al.*, "Algorithm and Data Optimization Techniques for Scaling to Massively Threaded Systems," in *Computer*, vol. 45, no. 8, pp. 26-32, August 2012.
- [8] Ma, X., Huang, P., Jin, X., Wang, P., Park, S., Shen, D. & Voelker, G. M. (2013). eDoctor: Automatically diagnosing abnormal battery drain issues on smartphones. In Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation.
- [9] S. K. Datta, C. Bonnet and N. Nikaein, "Android power management: Current and future trends," *2012 The First IEEE Workshop on Enabling Technologies for Smartphone and Internet of Things (ETSIoT)*, Seoul, 2012, pp. 48-53.
- [10] N. Nikzad, O. Chipara, and W. G. Griswold, "Ape: An annotation language and middleware for energy-efficient mobile application development," in *ICSE*, 2014, pp. 515-526.
- [11] N. Nikzad, M. Radi, O. Chipara, and W. G. Griswold, "Managing the Energy-Delay Tradeoff in Mobile Applications with Tempus," *In Proceedings of the 16th Annual Middleware Conference*, 2015, pp. 259-270, ACM.
- [12] S. Park, D. Kim, and H. Cha, "Reducing energy consumption of alarm-induced wake-ups on android smartphones," in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*. ACM, 2015, pp. 33-38.
- [13] D. Li, and W. G. Halfond, "An Investigation Into Energy-Saving Programming Practices For Android Smartphone App Development," *In Proceedings Of The 3rd International Workshop On Green And Sustainable Software*. ACM, 2014 pp. 46-53.
- [14] S. He, Y. Liu, and H. Zhou, "Demo: Optimizing Smartphone Power Consumption through Dynamic Resolution Scaling," *In Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, ACM, 2015, pp. 185-187.
- [15] M. A. Rumi, Asaduzzaman and D. M. H. Hasan, "CPU power consumption reduction in android smartphone," *2015 3rd International Conference on Green Energy and Technology (ICGET)*, Dhaka, 2015, pp. 1-6.
- [16] E. Ahmad and B. Shihada, "Green smartphone GPUs: Optimizing energy consumption using GPUFreq scaling governors," *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Abu Dhabi, 2015, pp. 740-747.
- [17] M. A. Shah, N. Naheed and S. Zhang, "Energy efficiency in smartphones: A survey on modern tools and techniques," *2015 21st International Conference on Automation and Computing (ICAC)*, Glasgow, 2015, pp. 1-6.
- [18] B. Zhao, W. Hu, Q. Zheng and G. Cao, "Energy-Aware Web Browsing on Smartphones," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 3, pp. 761-774, March 2015.
- [19] S. V. Rajaraman, M. Siekkinen and M. A. Hoque, "Energy consumption anatomy of live video streaming from a smartphone," *2014 IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC)*, Washington DC, 2014, pp. 2013-2017.
- [20] D. Li, Y. Lyu, J. Gui, and W. G. Halfond, "Automated Energy Optimization of HTTP Requests for Mobile Applications," *In Proceedings of the 38th International Conference on Software Engineering (ICSE)*, ACM, 2016, pp. 249-260.
- [21] S. K. Datta, C. Bonnet and N. Nikaein, "Self-adaptive battery and context aware mobile application development," *2014 International*

- Wireless Communications and Mobile Computing Conference (IWCMC)*, Nicosia, 2014, pp. 761-766.
- [22] F. Alam, P. R. Panda, N. Tripathi, N. Sharma and S. Narayan, "Energy optimization in Android applications through wakelock placement," *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, 2014, pp. 1-4.
- [23] S. Saha, S. Chatterjee, A. K. Gupta, I. Bhattacharya and T. Mondal, "TrackMe - a low power location tracking system using smart phone sensors," *2015 International Conference on Computing and Network Communications (CoCoNet)*, Trivandrum, 2015, pp. 457-464.
- [24] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proc. of the 2010 USENIX Technical Conference*, 2010, vol. 14.
- [25] G. Mei, and H. Tian, "Impact of data layouts on the efficiency of GPU-accelerated IDW interpolation," 2016, SpringerPlus, 5(1), p.1.
- [26] R. Strzodka, "Abstraction for AoS and SoA layout in C++," in *GPU Computing Gems: Jade Edition*, W. mei W. Hwu, Ed. Morgan Kaufmann, Sep. 2011.
- [27] Y. Y. Tseng, Y. H. Huang, B. C. C. Lai, and J. L. Lin, "Automatic Data Layout Transformation for Heterogeneous Many-Core Systems," in *Network and Parallel Computing*, pp. 208-219, 2014, Springer Berlin Heidelberg.
- [28] J. A. Stratton *et al.*, "Optimization and architecture effects on GPU computing workload performance," *2012 Innovative Parallel Computing (InPar)*, San Jose, CA, 2012, pp. 1-10.
- [29] P., and (Intel), S. M. "3D Vector Normalization Using 256-Bit Intel® Advanced Vector Extensions (Intel® AVX)", 2010, Retrieved May 09, 2016, from <https://software.intel.com/en-us/articles/3d-vector-normalization-using-256-bit-intel-advanced-ve>.
- [30] K. Kofler, B. Cosenza, and T. Fahringer, "Automatic Data Layout Optimizations for GPUs," in *Euro-Par 2015: Parallel Processing*, pp. 263-274, Springer Berlin Heidelberg.
- [31] P. y. Li, Q. h. Zhang, R. c. Zhao and H. n. Yu, "Data layout transformation for structure vectorization on SIMD architectures," *2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, Takamatsu, 2015, pp. 1-7.
- [32] H. Takizawa, S. Hirasawa, Y. Hayashi, R. Egawa and H. Kobayashi, "Xeolver: An XML-based code translation framework for supporting HPC application migration," *2014 21st International Conference on High Performance Computing (HiPC)*, Dona Paula, 2014, pp. 1-11.
- [33] T. Yamada, S. Hirasawa, H. Takizawa and H. Kobayashi, "A Case Study of User-Defined Code Transformations for Data Layout Optimizations," *2015 Third International Symposium on Computing and Networking (CANDAR)*, Sapporo, 2015, pp. 535-541.
- [34] C. Haine, O. Aumage, E. Petit, and D. Barthou, "Exploring and Evaluating Array Layout Restructuring for SIMDization," in *Languages and Compilers for Parallel Computing*, 2014, pp. 351-366, Springer International Publishing.
- [35] Intel® system studio," [Online]. Available: <https://software.intel.com/en->
- [36] J. Reinders, J. Je ers, and A. Sodani, *Intel Xeon Phi Processor High Performance Programming Knights Landing Edition*. Boston, MA, USA: Morgan Kaufmann Publishers Inc., 2016.
- [37] 2016, "Introduction to the clang AST — clang 4.0 documentation," 2007. [Online]. Available: <http://clang.llvm.org/docs/IntroductionToTheClangAST.html>. Accessed: Nov. 21, 2016.
- [38] 2016, "LibTooling — clang 4.0 documentation," 2007. [Online]. Available: <http://clang.llvm.org/docs/LibTooling.html>. Accessed: Nov. 21, 2016.
- [39] clang::CXXRecordDecl Class Reference," [Online]. Available: http://clang.llvm.org/doxygen/classclang_1_1CXXRecordDecl.html. Accessed: Nov. 21, 2016.
- [40] Getting Started with the LLVM System, *Getting Started with the LLVM System — LLVM 4.0 documentation*. [Online]. Available: <http://www.llvm.org/docs/GettingStarted.html>. [Accessed: 15-Dec-2016].
- [41] Canonical. (2016). Get Ubuntu. [Online]. Available: <https://www.ubuntu.com/download>. [Accessed: November 21, 2016].
- [42] Clang 4.0 documentation. [Online]. Available: <http://clang.llvm.org/docs/UsersManual.html>. [Accessed: November 21, 2016].
- [43] JNI tips,". [Online]. Available: <https://developer.android.com/training/articles/perf-jni.html>. Accessed: Nov. 21, 2016.
- [44] Mobile App Development | Visual Studio. [Online]. Available: <https://www.visualstudio.com/vs/mobile-app-development/>. Accessed :November 25, 2016.



Fatimah Abdulaziz Almusalli, received the B.A. degree in Computer science from college of science for girl in Dammam, King Faisal University in 2008. Currently, study Master in computer science at the College of Computer Science and Information Technology, King Faisal University, Saudi Arabia. Work as computer Trainer. Current job, IT administrator. her areas of interest include Software Engineering, Mobile Application Programming, Web Development, Network Management.



Dr. Noor Zaman acquired his degree in Engineering in 1998, and Master's in Computer Science at the University of Agriculture at Faisalabad in 2000. His academic achievements further extended with PhD in Information Technology at University Technology Petronas (UTP) Malaysia. He has vast experience of 16 years in the field of teaching and research. He is currently working as an Assistant Professor at the College of Computer Science and Information Technology, King Faisal University, Saudi Arabia since 2008. He has contributed well in King Faisal University for achieving ABET Accreditation twice, by working as an active member and Secretary for Accreditation and Quality cell for more than 08 years. He takes care of versatile operations including teaching, research activities, leading ERP projects, IT consultancy and IT management. He headed the department of IT, and administered the prometric center in the Institute of Business and Technology (BIZTEK), in Karachi Pakistan. He has worked as a consultant for Network and Server Management remotely in Apex Canada USA base Software house and call center. Dr. Noor Zaman has authored several research papers in indexed journals\international conferences, and edited six international reputed Computer Science area books, has many publications to his credit. He is an associate Editor, Regional Editor and reviewer for reputed international journals and conferences around the world. He has completed several international research grants\ funded projects and currently involved in different courtyers. His areas of interest include Wireless Sensor Network (WSN), Internet of Things IoT, Mobile Application Programming, Ad hoc Networks, Cloud Computing, Big Data, Mobile Computing, and Software Engineering.



Raihan ur Rasool is a Fulbright Alumnus of the University of Chicago, USA. He is currently affiliated with Victoria University, Melbourne. His research interests include large-scale systems, security and computer architecture. His research work, comprising over 42 articles is published in various international conferences and journals. He has published in leading avenues of research like ISCA, HiPEC, CCGrid, ACM SIGARCH, IEEE Transactions on Cloud Computing, JNCA & FGCS.