International Journal of Advanced Smart Convergence Vol.6 No.1 26-31 (2017)
https://doi.org/10.7236/IJASC.2017.6.1.26

IJASC 17-1-4

A Case Study of a Navigator Optimization Process

Doosan Cho

Electrical & Electronic Engineering, Sunchon National University, Korea
dscho@scnu.ac.kr

Abstract

When mobile navigator device accesses data randomly, the cache memory performance is rapidly deteriorated due to low memory access locality. For instance, GPS (General Positioning System) of navigator program for automobiles or drones, that are currently in common use, uses data from 32 satellites and computes current position of a receiver. This computation of positioning is the major part of GPS which accounts more than 50% computation in the program. In this computation task, the satellite signals are received in real time and stored in buffer memories. At this task, since necessary data cannot be sequentially stored, the data is read and used at random. This data accessing patterns are generated randomly, thus, memory system performance is worse by low data locality. As a result, it is difficult to process data in real time due to low data localization. Improving the low memory access locality inherited on the algorithms of conventional communication applications requires a certain optimization technique to solve this problem. In this study, we try to do optimizations with data and memory to improve the locality problem. In experiment, we show that our case study can improve processing speed of core computation and improve our overall system performance by 14%.

Keywords: *Optimization, Low Power, Embedded System, Data Memory, Performance*

1. Introduction

A real-time embedded system for normal communication applications has a small-sized hardware cache (2 to 16 KB), and randomly accesses data of a size more than several hundred times larger. As a result, cache memory utilization (hit-ratio) is low, making it difficult to develop a successful system [1]. In this study, we aim to improve system performance by solving low data locality problem for GPS applications of navigators.

Today, many embedded systems are designed and built with limited hardware resources. Instead of lowering the hardware cost of the system in order to increase price competitiveness, the burden of meeting the required performance criterion is on a software system. One of the most important aspects of improving performance through software optimization is optimizing memory-related modules, and the most effective technique is to increase hardware cache utilization. The proposed method is a new approach and differs from

Manuscript Received: Feb. 15, 2017 / Revised: Feb. 25, 2017 / Accepted: Mar. 6, 2017
Corresponding Author: dscho@scnu.ac.kr
Tel: +82-61-750-3577, Fax: +82-61-750-3570
Department of Electrical & Electronic Engineering, Sunchon National University, Korea

existing methods. The techniques we tried in this study are optimized for GPS applications and are specifically applied to a certain memory system. Therefore, more performance improvement than existing techniques can be obtained. It is developed specifically for the application and memory hierarchy.

2. Background

Car or drone navigation refers to an advanced transportation system that reduces traffic congestion and creates a safe and pleasant driving environment by collecting, analyzing, and providing information on the current location of the vehicle, the traffic situation, and the traffic situation information of the driver. This systems are closely related to ITS (Intelligent Transportation Systems) and are a part of the ITS system. In the early stage of ITS technology development, there has been no direct technology development related to drone/car navigation as well as related fields. Recently, researches on the navigation focusing on vehicles have been actively conducted. A navigation is a combination of GPS technology and mobile communication technology. Its information provision service is a field that can overcome limitations of ITS related services that are difficult to provide a wide range of services temporarily, and actual technology development and commercialization are actively performed.

Since the internal system of the navigation terminal is composed of the main body, the wireless transmission/reception system (GPS, wireless modem part, other wireless transmission/reception part), navigation part, peripheral device, etc. The main role of the terminal module is as follows.

O Body: Controlling S/W of each module

O Wireless transmit/receive system

- GPS: Vehicle location confirmation (route search and emergency location notification)

- Wireless modem part: Transmitter role (send/receive information from service center)

- Other wireless transmission/reception: receiver role (traffic information acquisition)

O Navigation part: Path search light (display current position and route)

O Peripherals: Multimedia, MP3, external interface, etc.

Fundamentally, GPS uses the principle of triangulation. In a typical triangulation, the location of an unknown point is determined by measuring the size of the two angles and the length of the sides, except for that point. Determining the location of an unknown point by measuring the length of two sides is the difference from the classic triangulation. The distance from the satellite to the receiver is calculated by measuring the time difference between the point of time of the sign signal generated at each satellite and the point of time of reception, and then multiplying by the speed of light (distance = speed of light × elapsed time). In order to determine the position of the receiver based on the position of the satellite, it is necessary to know the exact position of the satellite in addition to the distance data. To calculate the position of the satellite, a trajectory force transmitted from the GPS satellite is used.

To calculate the position of a user on three dimensions, we need to determine three unknowns x, y, z mathematically, and we need to get signals from at least three satellites because we need three equations. However, to measure the elapsed time to calculate the distance between the GPS satellite and the user, the time must be synchronized with the satellite and the receiving period. Multiplying the speed of light with a small time error will result in a great distance error. However, satellite clocks have very accurate atomic

clocks, but because receivers use cheap clocks, it is virtually impossible to physically synchronize the two clocks exactly. This problem is mathematically overcome by the receiver. That is, the receiver computes x, y, and z up to time t. T to synchronize the satellite clock with the receiver clock. Here, the number of unknowns to be determined is increased to four signal of satellite, and equations of four or more are required. In other words, it is necessary to receive signals from at least four or more satellites in order to accurately calculate the position of the user.

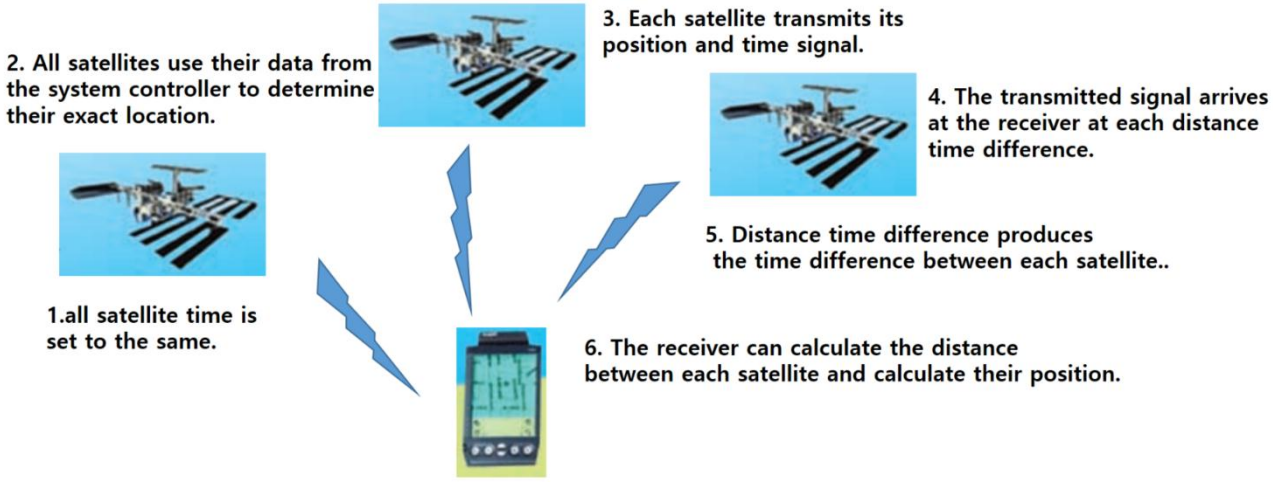

Figure 1. The workflow of GPS

3. Platform

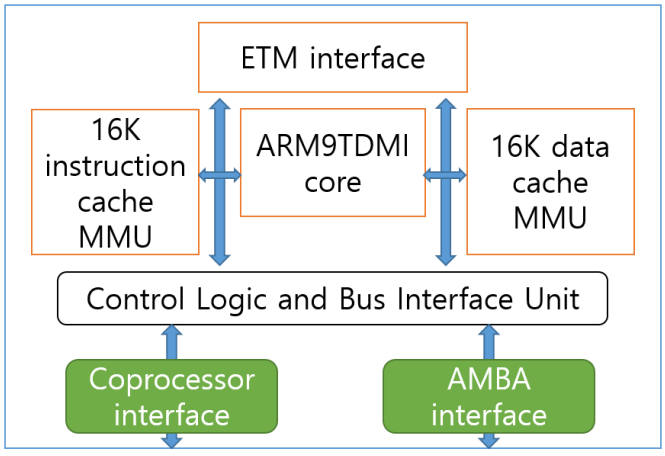

Figure 2. ARM920T Architecture

```
foo(){
    for(i = 0 ; i < n ; i ++)
        print map[rand*65536]
```

Figure 3. An example of main code

ARM920T is used as our target system platform. The ARM9 is a 5-stage integer pipeline, with I-CACHE and D-CACHE of Harvard architecture, each 16KB. Unlike the von Neumann architecture, the Harvard architecture is more advantageous for certain applications because the cache memory is divided into I-cache and D-cache. Figure 2 shows the ARM920T diagram. The OS uses WinCE 6.0 and MS ARM9 compiler. ARM9 shares the same instruction set with ARM7, and this uses TLB for each cache. It operates with 5 stage pipelined structure. Figure 3 shows the major computation part of positioning. The array map*[rand*65536]* represents random memory access in the computation. The program used in this case study is a car GPS application that accesses data of 256KB size in a random pattern. The data used at one time is 16 times larger than the cache size. Figure 3 shows the main code used for memory accesses.

In the ARM architecture, cache memory is very important. A cache is one that allows faster processing by a faster buffer memory in the middle to access slower memory [2]. The faster memory is called the cache memory. Depending on hierarchy, capacity, and operation policy of ARM architecture, system performance may vary by as much as 200%.

3. Locality Analysis of Data Accesses

To improve cache performance, we first need to determine the correlation between data access locality and performance. To understand this, we used '%' for (modulo operation) with an industry code and real navigator system board [3][4]. That is, the locality of the data used is artificially increased, and the degree of performance improvement possible is confirmed.

Figure 4 shows the code with modulo '%' operation. In the experiment, each effect was measured by changing the random access coverage size of m to 3,10,100,1000,5000. If the modulo '%' operation is not applied, the processing time requires 1.67 seconds. This means that the current GPS system can process 1.67 seconds of data for 1 second. In order to operate the implemented GPS system normally, the performance should be 3 seconds or more. In conclusion, in order to achieve the target performance, it is necessary to obtain a localization level of "modulo 3" or higher, which is impossible with a single technique, since it cannot reduce data coverage to 3 from 65536.

```
foo(){
    for(i = 0 ; i < n ; i ++)
        print map[(rand*65536)%m] }
```

Figure 4. An optimization with a modulo operation

Table 1. Footprint of data processing throughput by increasing Modulation size

Mod operation with 3	2.66second(data processing throughput)
Mod operation with 10	2.68
Mod operation with 100	2.62
Mod operation with 1000	2.38
Mod operation with 5000	2.02
No Mod operation	1.67

4. Locality improvement by data compressing

In order to obtain the required performance at a lower cost than the algorithm modification of the GPS application, the data is compressed. By reducing the size of the data, it is possible to improve the locality and improve the performance of the cache system.

Lossless technique is required for data compression. In the case of lossy compression, the total cost of performance can be reduced due to the increase in the amount of recovery computation. The proposed technique is as follows.

The data used in normal communication applications is composed of tables (ex, routing table, ip table, location table, etc.). A table can be divided into two fields, which can be divided into an address field and a value field that contain specific data. In case of GPS code, it uses the table that searches the location based on the data received from the satellite and uses the encoded address when searching for the location. Therefore, when the address field and the data field are classified and the value is tested, it can be confirmed that there are many overlapping parts. It is a phenomenon that occurs due to the characteristic of searching a position obtained by receiving various index values from various satellites. This phenomenon is called value locality [5]. In this case, it is possible to compress the entire data size by separating the address and value tables.

For example, in the case of the GPS code, it is confirmed that the table used is composed of 617 actually divided values among 65535 data. Therefore, the data table can be divided into two 16-bit address tables (Address Table: 65536 2byte array, 128kbyte) and 32bit data table (617 4byte array, 2.5kbyte). The total capacity of both data is 130.5kbytes, which can be reduced by about 1/2.

```
foo(){
    for(i = 0 ; i < n ; i ++)
        print data[address[rand*65536]] }
```

Figure 5. An optimization code (with 130KB table)

As shown in Figure 4, the memory accesses are increased one time in the address table and one time in the value table. However, since the size of the total data is reduced by half, the locality is increased. An average increase of 14% from 1.67 seconds to 1.87 seconds. One of the applied techniques has the biggest improvement effect.

5. Conclusion

In a real-time embedded system development, a key part in determining system performance is related to memory access latency [6][7]. In design step, this problem can be changed by hardware, but it is not easy to improve the memory system performance at the time when the hardware part development is completed. As a result, various compiler optimization techniques can be used to improve software performance, but there is a limit to performance improvement without the support of an algorithm side. In this study, we analyze the characteristics of the data used as the input of the target application and experimentally show that the performance of the whole system can be improved by reconstructing the data to improve the locality.

Acknowledgement

This research was supported by Unmanned Vehicles Advanced Core Technology Research and Development Program through the National Research Foundation of Korea(NRF), Unmanned Vehicle Advanced Research Center(UVARC) funded by the Ministry of Science, ICT and Future Planning, the Republic of Korea. (No. 2016M1B3A1A03937725).

References

[1] Wm. A. Wulf and Sally A. McKee, "Hitting the Memory Wall: Implications of the Obvious," ACM SIGARCH Computer Architecture News, vol.23, no.1, pp.20-24, Mar. 1995.

[2] John L. Hennessy and David A. Patterson, "COMPUTER ARCHITECTURE A Quantiative Approach," 5th edition, Morgan Kaufmann, pp.72-78, Sep. 2011.

[3] ARM920T Technical Reference Manual, 1 edition, 2001.
[online] http://www.atmel.com/Images/ARM_920T_TRM.pdf

[4] K. Hazelwood, and A. Klauser, "A dynamic binary instrumentation engine for the ARM architecture," Compilers, Architecture and Synthesis for Embedded Systems Conference, Oct. 2006.

[5] Jun Yang, Rajiv Gupta, “Frequent value locality and its applications,” ACM TECS, 2002.

[6] Carr, S., McKinley, K. S., & Tseng, C. W., “Compiler Optimizations for Improving Data Locality,” ACM SIGPLAN Notices, 29(11), 252-262, 1994.

[7] Kathryn S. Mckinley, Steve Carr, Chau Wen Tseng, “Improving Data Locality with Loop Transformations,” ACM Transactions on Programming Languages and Systems 18(4), 424-453, 1996.