International Journal of Computational Intelligence Research
ISSN 0973-1873 Volume 13, Number 5 (2017), pp. 815-823

http://www.ripublication.com

Advanced Specialized Processor Architecture for Samrtphones

Rohit Kumar[1*], Rohit Bajaj[1], Vijay Kumar Singh[1], Kshitiza Vasudeva[1], Lokesh Pawar[1]

[1]Chandiharh University, Gharuan, Mohali, Punjab, India.
**Corresponding Author*

Abstract

Smartphones have become essential part of our daily life. These devices now a days are not just communicative device but a powerful computer. Using these devices a large number of computationally intensive tasks can be executed with ease which were not possible. These samrtphones uses ARM (Advanced Risk Machines) based processor, which provides reasonable energy efficiency and computational power. As the smartphones are battery powered devices, energy efficiency remains a key concern and is a serious challenge for the designers. In this research paper a novel hardware architecture has been proposed which alongside increasing computational power will provide huge energy savings. The architecture leverages the problem of dark silicon to provide an energy efficient solution for the samrtphones.

Keywords: Smartphone development issues, smartphone processor architecture, specialized processor architecture, hardware.

I. INTRODUCTION

The processor architecture of conventional processor is not suitable for smartphones as conventional processor consumes very high energy. We need some specialized processor architecture to work with energy starved "samrtphones". This is what that has been discovered in this research paper The problem of utilization wall, dark silicon, (discussed later) hardware reconfiguration with code change has been the inspiring factors for carrying out this research work. All of these have been major factors now a days for all type of microprocessor based electronic projects and products which requires extensive and real time and mobile computational power. Beside this efforts have been made to reduce the communication overhead in major

sections of code by designing specialized hardware and to reduce the fetch decode cycle by large extent. It has been proved in recent researches that reducing these two types of communication delays and by tackling the problem of dark silicon will improve power efficiency by 7x to 1000x [1-2].

This research work has tried to leverage the problem of dark silicon to attain efficiency and prolonged battery life by introducing an efficient and powerful processor. Focus has been laid on reduction in communication delays in processor design. The proposed processor design and architecture fulfills the research objective.

II. APPROACH USED FOR DEVELOPMENT

Top down approach has been used for the design of the concerned hardware. Initially broader architecture has been proposed, which is later refined and elaborated in detail. When the real manufacturing will be done the reverse methodology need to be followed although the real manufacturing details are out of scope from this research work. Before introducing the architecture of the system one must know the terms namely utilization walls and dark silicon which form the base of the research carried out [3-5].

2.1 Utilization Wall

The concept of utilization states that with each new process generation the number of transistor that can be switched on full frequency drops exponentially due to power constraints. Due to this a large portion of chip area is not utilized. So a narrow wall is created that separates the used and unused portion of the silicon chip, this separation wall or boundary is called utilization wall. According to Moore's law the number of transistor grows exponentially with each passing year. This law has been true from the past many decades but due to power constraints people are not able to operate this transistor count at full frequency which results in under utilization. This problem of under utilization has been tackled in this research work.

In 1965 Gordon E. Moore, presented a law, this law is the observation that over the history of computing hardware, the number of transistors on integrated circuits doubles approximately every two years. But since the breakdown of Dinnard theory [7] this law no longer holds for latest ultra scale integrated circuits. Dennard's scaling rules observe that voltage and current should be proportional to the linear dimensions of a transistor, implying that power consumption (the product of voltage and current) will be proportional to the area of a transistor. This property implies that shrunk MOSFETs, CMOS will consume less power, and formed the basis of Moore's Law. But this scaling theory had failed and led to multi-core architectural designs for processors. It happened because the power resources remained same and are not able to switch the chip at full frequency [6-9].

2.2 Dark Silicon

The portion of silicon chip that can-not be utilized due to some reason is called dark silicon. Dark silicon can appear for different reasons e.g. if a chip is not programmed properly or lack of energy to switch transistors. The existence of dark silicon due to power constraints has been the factor studied and resolved in this research work. In addition to this specialized conservative has been presented which has tried to eliminate the problem of 'Dark Silicon' [7].

III. Proposed Architecture

Figure 1.1 presents the proposed architecture. Each of the major block and components in the proposed architecture has been explained in detail in rest of the chapter.

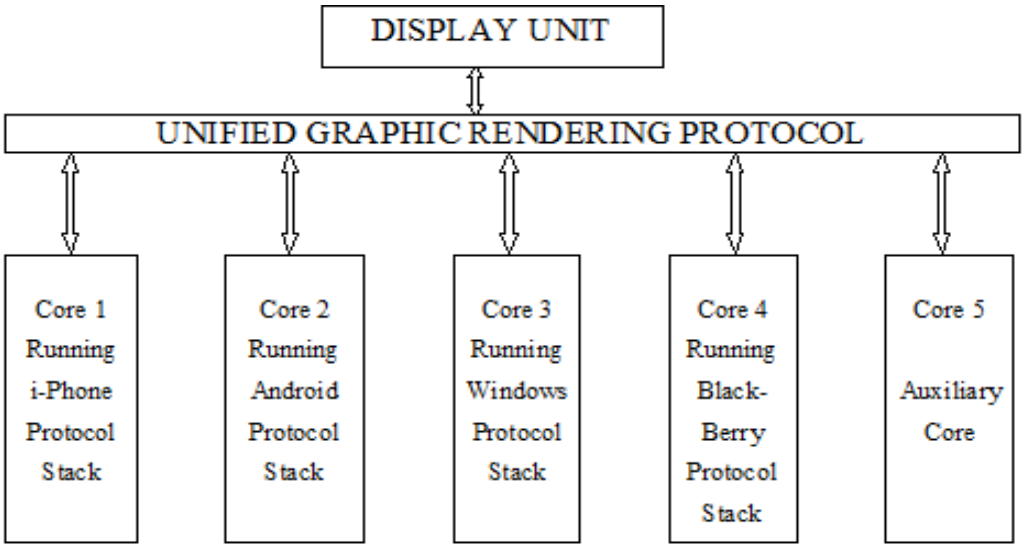

Figure 1.1.: Architecture Running multiple protocol stacks.

3.1 Cores/Processors and Corresponding Protocol Stack

Each of the major core is an independent multi-core processor with many number of energy aware conservative cores. Each of the cores will run a Specific Mobile OS. e.g. the first core is running Android Protocol Stack, other cores are running Symbian, Blackberry, Windows and respectively and one core reserved for future use. Each core is tailored according to the OS to be used. The conservative cores inside each core can configured for any specific code. Conservative cores are used to run some specialized piece of code and the code that also runs frequently. Also, the cores can be reconfigured as the code changes. Each core/processor has conservative cores of various sizes to best fit the code of various sizes and complexity.

3.2 Unified Graphic Rendering Protocol

Each of the mobile platform has its own graphic rendering mechanism, so one need to have a common graphic rendering mechanism that will adapt according to the mobile platform used (i.e. the core being used or when multiple cores are being used).

Since multiple core executes multiple software stacks and each software stack has its own graphic rendering mechanism, so each must be implemented carefully and specifically. With this specialized hardware is used for implementing graphic rendering and graphic library functions for all platforms. This increases the complexity of the problem. In the presented model situation is more grave as multiple graphic rendering mechanisms need to be implemented and each needs its own specialized hardware as well. Actual implementation of graphic rendering mechanism is outside the scope of this research work.

3.3 Architecture of Single Core

Figure 1.2 describes the detailed architecture of each core used in the proposed model (Figure 1.1). Each core consists of array of tiles. Each tile of the processor is a full-fledged processor with a set of conservative cores. Number of conservative core in a processor can vary from 8 to 90. Each tile implements code specific to one or set of applications which are closely related and each conservative core within each tile is supposed to run some specialized code from the concerned applications. Tiles are chosen according to the application type and requirement. If an application has many modules then tile with large number of cores will be preferable and will be chosen automatically by the application.

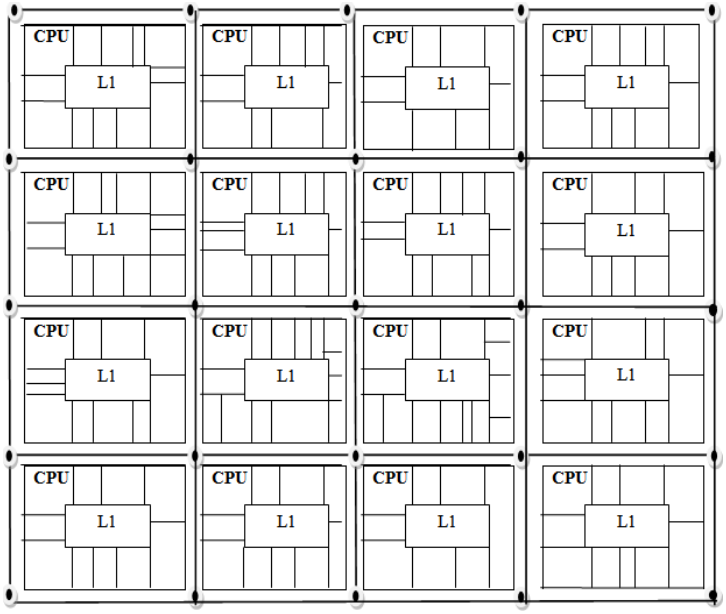

Figure 1.2.: Array of 16 processing units.

Figure 1.3 shows components of a single tile and each of these tile is a full fledged processor in itself. Each tile has on chip network (OCN), instruction cache, data cache and number of conservative cores. Size of cores and number of cores will vary from tile to tile. All conservative cores are tightly connected with the data cache for communication.

C-cores featured here are size variant and comes for different sized codes. The data cache or L1 is tightly integrated with the cores and provides very tight integration among them. All sort of data communication is done through this data cache. Since a single core has sixteen different tiles in a core and each core also have sixteen different L1 data cache and all of these caches must be kept consistent with each other. To bring consistency directory based cache coherence protocols have been used.

When the full system executes only few cores are active in execution at any moment of time so the problem of utilization wall is contained. Due to specialized nature of conservative cores these c-cores consumes very little energy which has been utilized in the presented model.

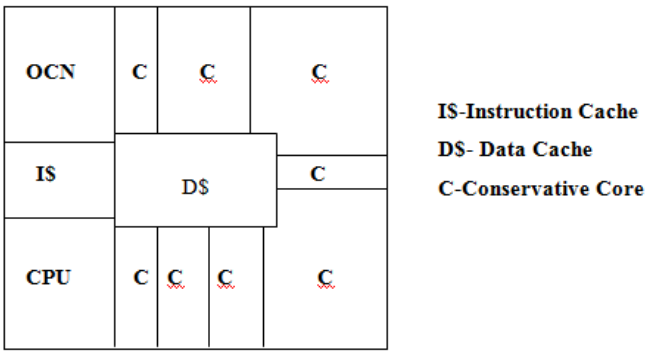

Figure 1.3.: Component of each tile

A translation look-aside buffer (TLB) is a cache memory management hardware uses to improve virtual address translation speed. All current desktop, notebook, smartphones and server processors use a TLB to map virtual and physical address spaces, and it is nearly always present in any hardware which utilizes virtual memory. Each tile in the architecture has its dedicated TLB.

Figure 1.4 presents the detailed architecture of each tile. It has been observed during trials that c-cores consume 18 times less energy on average than conventional processor. C-cores comprise most of the execution area so only very small code executes on general CPU. Power gating and clock reduction techniques are used to minimize the power consumption. Power gating disconnects the cores those are not used in present computation to save lots of energy. Clock reduction is used to reduce power consumption when only is being used for computation. When large code is executed clock frequency is increased for efficient computations.

3.4 Cache Coherence

Each tile or node shares its cache memory with other tiles this leads to the problem of cache coherence. Cache coherence implies that each node or tile must be given updated copy of the data modified in other cache. If an old value will be given to the requesting node then it will result in inconsistent computation and thus incorrect information will be represented by the system. In the present model directory based cache coherence protocol has been used. In this protocol each node maintains a directory of the items under its cache memory in the form of bit vectors. If a request comes and node contains updated copy of the item then it responds with the data item. But in case if the data item has been modified by any other node than it forwards the request to that node and modifies its own cache as well.

3.5 Conservative Cores

As already mentioned each tile has conservative cores of various sizes and each chip contains 8 to 90 conservative cores. Idea behind keeping different sized core is to use them for different sized code modules. Small code will be placed on small cores and big cores will be used for larger code.

Automated tool exists that finds out the appropriate cores for different application. These cores can also be reconfigured to accommodate code changes and new applications can be installed on them with ease.

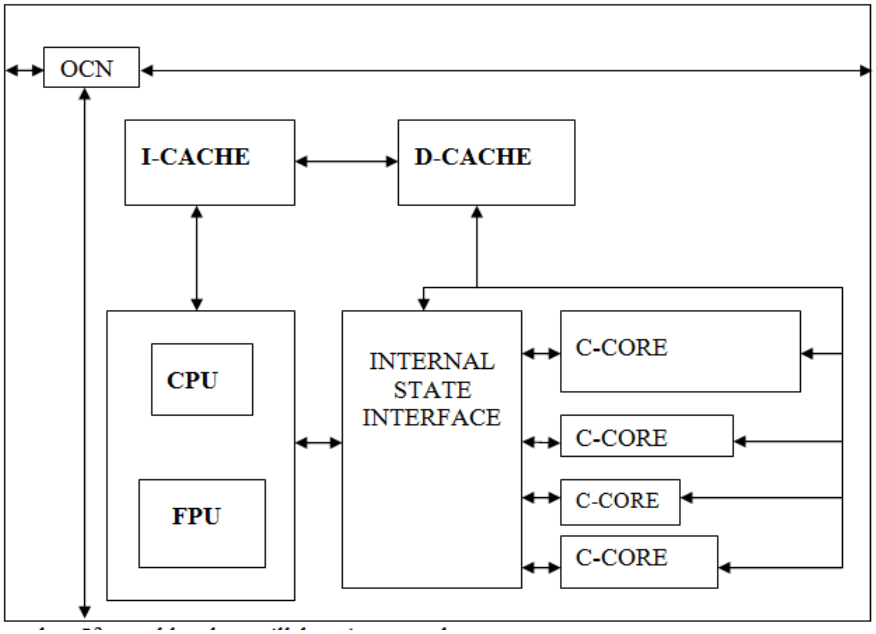

Figure 1.4.: Architecture of Each Tile.

3.6 Fine Grained Conservative Cores

When many small sized c-cores exist in the system then the system is said to be a fine grained CPU and has been used in the proposed architecture. By the use of fine grained c-cores majority of the system code can be synthesized in c-cores and during execution c-cores span up to 90 % of execution time. Use of extremely large number of c-cores is not recommended and is not possible either as they increases the communication load on the processor and performance degradation occurs. This problem is also known as multi-processor scaling problem. Due to this the processor chip area must not be divided into very large number of c-cores.

IV. ENERGY SAVINGS ACHIEVED BY THE USE OF C-CORES

It is the use of conservative cores that brings lots of energy savings. Conservative cores contains hardwired synthesized version of the software code. Due to hardwired implementation of the code, fetch-decode cycle for executing instruction is eliminated, data path communication overhead is also eliminated and there is no requirement of register file and instruction cache at all, as all the operands and data are directly embedded into hardwired logic except data supplied by user while executing an application.

Table 1.1 presents the energy savings obtained by the use of conservative cores. Conservative cores just need to interact with data cache which amounts to only six percent of the total energy consumed by the baseline CPU. In contrary to it conventional baseline CPU consumes lots of energy in fetch-decode cycle which amounts to 19 % of energy consumption, maximum energy is wasted in data path communication which consumes 38 % of the energy, with this instruction cache consumes lots of energy amounting to 23 %, followed by register file with 14 % percent energy consumption.

In conservative cores energy is consumed by data cache only, rest of the factors has been eliminated in conservative cores which results in very high energy savings and these energy savings needs to be capitalized to make a very efficient processor design. In this research work these saving have been capitalized to make the proposed processor prototype and to achieve the desired objectives.

Table 1.1.: Energy Consumption Comparison in Baseline CPU and C-cores.

Factors	**Energy consumed in Baseline CPU (%)**	**Energy Consumed in C-Core (%)**
D-Cache	6	6
I-Cache	23	NIL
Fetch/Decode	19	NIL
Register File	14	NIL
Data Path	38	NIL

4.1 C-core Mimics Software Code

C-cores actually mimics the software code that is synthesized on it. During synthesis all of the instruction given in the program are hardwired in c-cores with all associated operands. E.g Consider the following for loop:

```
for ( i=0; i<=10; i++)
  { a=a+i;
  }
```

Figure 1.5 presents an abstract hardware synthesis of the for loop. The variable (i) used in the loop has taken the shape of a fixed register of four bits, and will be initialized with 0, the comparison in the for loop has been replaced by a fixed logic comparator, addition had been implemented by the adder circuit, the parenthesis are implied by the arrow among the for loop components. One more register names (a) will be created to hold the value of variable (a). The number of bits taken by the (a) register will be determined by the initial value of (a) and plus execution of for loop.

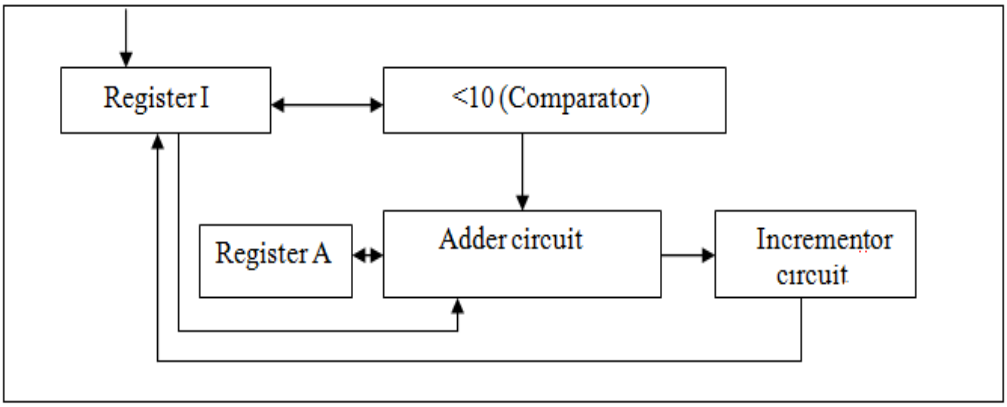

Figure 1.5 Synthesis of for loop in c-core.

V. CONCLUSION

The proposed prototype has been designed to obtain lots of energy savings over conventional processor design. The proposed prototype efficiently reduces the impact of dark silicon and utilization wall. Conservative cores have been introduced which are actually responsible for removing the problem of utilization wall and dark silicon.

Each processor tile contains variant number of conservative cores and has all other circuitry like general CPU, instruction cache and data cache etc. for executing any kind of software code. Identification of hot code is must for achieving good efficiency and power savings. Automated tools must have to be used as manual synthesis is not

possible for large code bases. Work is being done on optimization of automated tools to gain maximums efficiency.

Actual synthesis is also very difficult as the prototype needs very large area. The tools like verilog must be programmed carefully and must be thoroughly tested for good synthesis.

REFERENCES

[1] Mombert, G., http://www.digitaltrends.com/mobile/what-is-asmartphone/ [online], last *seen* dec, 2010.

[2] Johnny John and Chris Riddle, "Smartphone Power*", proceedings of DAC, Anaheim, California, USA*, pp. 935-936, 2010.

[3] Vinay Mehta, http://berylsystems.com/smartphone.pdf [online], *seen* oct, 2010.

[4] Steven Cavanagh and Yingxu Wang, "*Design* of a Real-Time Virtual Machine (RTVM)", *proceedings of Electrical and Computer Engineering 2005 Canadian Conference,* pp. 2021-2024, May, 2005.

[5] Omar A. Fres and Ignacio G. Alonso, "Rovim: A Generic and Extensible Virtual Machine for Mobile Robots", *Fifth International Conference on Systems held in USA*, pp. 37-40, 2010.

[6] Michael Bedford Taylor, "The Raw Microprocessor: A Computational Fabric For Software Circuits and General-Purpose Programs", *IEEE proceedings*, pp. 24-35, 2002.

[7] Robert H. Dennard, "Design of lon-Implanted MOSFET's with Very Small Physical Dimension*", IEEE Journal of Solid-State Circuits*, vol. SC-9, pp. 256-268, October 1974.

[8] [8]. Asaf Ashkenazia, Dimitry Akselrodb and Yossi Amona, *"*Platform Independent Overall Security Architecture in Multi-Processor System-on-Chip ICs for Use in Mobile Phones and Handheld Devices", *proceedings of World Automation Congress (WAC*), vol.33, issue no.5-6, pp. 407-424, 2007.

[9] [9]. Meira Levy, Peretz Shoval, Bracha Shapira, Aviram Dayan and Meytal Tubi, "Task Modeling Infrastructure for Analyzing Smartphone Usage" *Ninth International Conference on Mobile Business / 2010 Ninth Global Mobility Roundtable*, pp.264-271, 2010.