Matrix Academic International Online Journal Of Engineering And Technology
ISSN: 2348-3326, SJIF 2013 3.9
www.maioj.org © MAIOJET
Volume 4, Issue 2

Advanced Tools and Techniques for Re-configurable Processor Architectures

Rohit Kumar[#]

[#]*Department of Computer Sc. & Engineering, Chandigarh University, Gharuan, Mohali.*

rohitkumar.cse@cumail.in

Abstract— There have been tremendous advancements in the processor architectures and tools used to manufacture and reconfigure them. The problem of utilization wall, dark silicon, hardware reconfiguration with code change has been the inspiring factors for advancement in this field. 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 000x.

***Keywords*— Processor, CPU, Protocols, Algorithms, Energy.**

I. INTRODUCTION

A. 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 thesis. In addition to this specialized conservative has been presented which has tried to eliminate the problem of 'Dark Silicon' [-5].

B. 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 thesis work.

In 965 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. It has been observed that Although a fixed-size chip's computing capabilities continue to increase exponentially at 2.8x per process generation owing to both increases in maximum transistor count (2.x) and improved transistor frequencies (.4x), the underlying energy efficiency of the transistors is only improving at a rate of aout .4x. Because they must adhere to exploit these improved capabilities to the extent they are matched by an equivalent improvement in energy efficiency. The short-fall of 2x per generation is the cause of the utilization wall, and leads to the exponentially worsening problem of dark silicon [6-4].

II. BEHAVIOR AND ADVANCED TOOLS

A. 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<=0; i++)
   { a=a+i;
   }
```

Figure 1 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 [5].

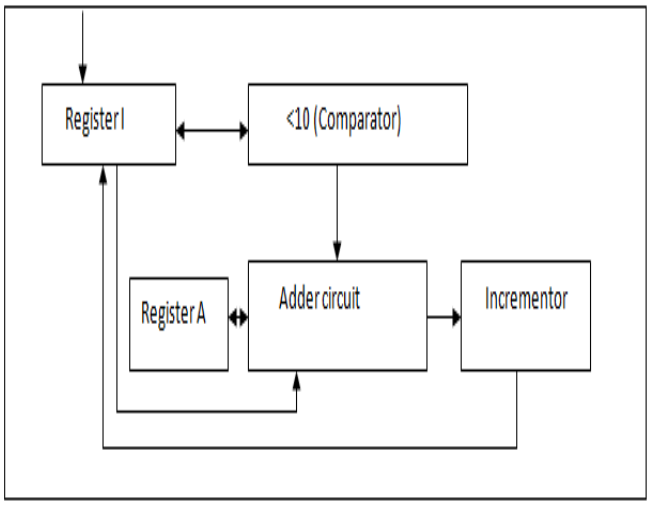

Fig. Synthesis of for loop in c-core.

B. Execution Model

Automated tools are used to profile work load. At design time, automated tool clusters c-cores on the basis of profiling of typical smartphone work-loads, examining both control flow and data movement between code regions. It places related c-cores on the same or nearby tiles, and in some cases, replicates them. At runtime, an application starts on one of the general purpose CPUs, and whenever the CPU enters a hot-code region, transfers execution to the appropriate c-core. Execution moves from tile to tile on the basis of the applications that are currently active and the c-cores they use. Coherent caches let data be automatically pulled to where it's needed, but data associated with a given c-core will generally stay in that c-core's L cache.

C. Synthesis of Android stack on a Core

Android is an open-source mobile software stack developed by Google that features a Linux kernel, a set of application libraries, and a virtual machine called Dalvik. User applications, such as those available in the application store, run on top of the Dalvik virtual machine [6]. It has been found that Android is well-suited for c-cores for several reasons. First, although many applications are available for download, Android has a core set of commonly used applications, such as a Web browser, an e-mail client, and media players. Typically, hot code is concentrated in the application libraries, the Dalvik virtual machine, and a few locations in the kernel. Because the hot code is well concentrated, targeting all these components with c-cores lets user attain high coverage over the source base and a significant impact on overall energy usage. Although c-cores support patching, which reduces the impact of post-silicon source base modification, users are also aided by smartphones' short replacement cycle (typically every 2 to 3 years), which lets smartphone chip designers deploy new c-cores to target new applications. The c-cores interface lets Android phone designers remove c-cores from their designs without impacting code compatibility [7-8].

D. Patching Support

To remain useful as new versions of the any platform emerge, the proposed architectures c-cores must adapt accordingly. To support this, c-cores include targeted re-configurability that lets them maintain perfect fidelity to source code, even as the source code changes. The adaptation mechanisms include redefining compile time constants in hardware and a general exception mechanism that lets c-cores transfer control back and forth to the general-purpose core during any control flow transition. Adding this re-configurability increases the energy and area needs for c-cores, but significantly improves the span of years over which c-cores can provide energy savings.

E. Implementing Android Graphic Library

Figure 2 shows a tile used to implement graphic function for Android based phone. The tile contains nine c-cores targeting important functions from the Android code base. Of these nine targeted funcions, seven come from libkia, a 2D graphics library that provides compositing, rendering, and geometry calculations for most Android applications. The other two come from a JPEG decompression library and a fast Fourier transform (FFT).

Fig. 2 Single Tile targeting Android Graphic Library Functions.

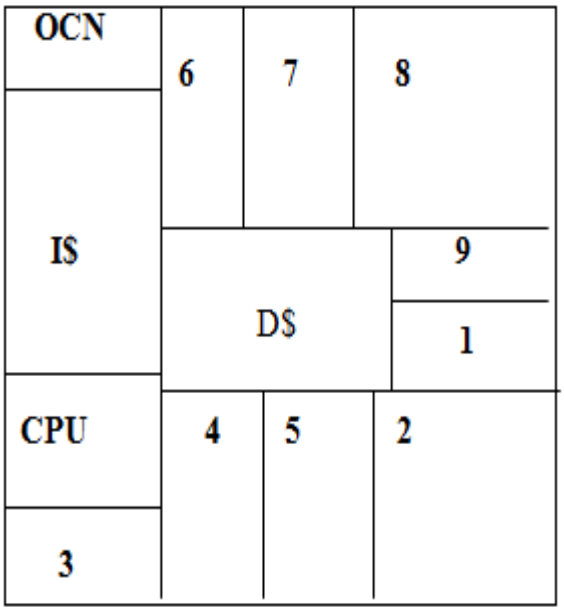

TABLE . Size and dynamic execution coverage of C-Cores for Functions.

No.	Description	Android function	Dynamic execution coverage (%)	No. of static instructions	Size (mm^2)
1	Dithering function	S32A_D565_Opaque_Dither	2. 78	80	0.55
2.	Down sampling	S32_opaque_D32_filter_DXDY	2.20	86	0.07 0
3.	Bit-block image transfer subroutine	S32A_Opaque_BlitRow32	.5	96	0. 024
4.	Render with overlay	Sprite_D6_S4444_Opaque::blitRect	.5	96	0. 059
5.	Saturating down sampling	Clamp_S6_D6_filter_DX_shaderproc	0.80	97	0.06 3
6.	Fast Fourier transform (FFT)	fft_rx4_long	0.76	38	0.06 6

7.	Image conversion algorithm	ycc_rgba_8888_convert	0.6	92	0. 046
8.	Lempel-Ziv decompression routine for GIF files	DGifDecompressLine	0. 59	334	0. 68
9.	Image format conversion for dithering	Sample_Index_D565_D	0.57	67	0. 032

Table 1 presents basic image processing functions for Android software stack. The tile in Figure 2 contains c-cores for these nine functions, many of which are from libskia, Android's 2D graphics library. The c-cores range from 0.024 mm^2 to 0.68 mm^2 and cover 0 to 90 percent of execution. Other tiles will have c-cores for other Android functions.

Most of the graphic library can be implemented with a single tile and is sufficient for graphic rendering. If the library contains many modules then tile with more number of cores must be used. Even if the graphic library is too large then more than one tile can also be used [9].

F. Energy Consumption Per Instruction

On average, each c-core occupies 0.064 mm^2 and runs at ,568 MHz. Together, all the c-cores occupy 58 percent of the tile's area. The code they execute accounts for a total of 0.6 to 89.9 percent of execution for the benchmarks. Each of the core has 6 tiles and executes a full protocol stack with associated applications. Figure 3 (a,b) shows the projected energy savings in processor and the origin of these savings. The savings as mention already come from two sources. First, c-cores don't require instruction fetch, instruction decode, a conventional register file, or any of the associated structures. Removing these reduces energy consumption by 56 percent. The second source of savings (35 percent of energy) comes from the specialization of the c-cores' data path.

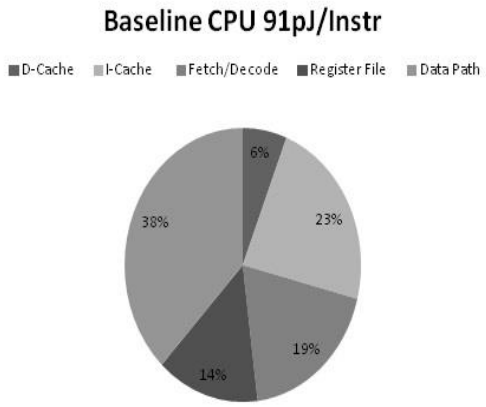

Fig. 3(a) Power consumption

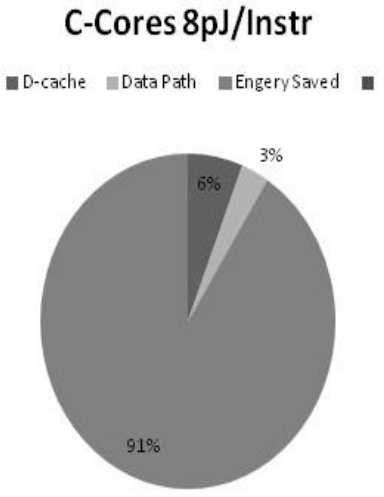

Fig. 3(b) Power Saving.

The energy saved needs to be capitalized and conservatives cores effectively utilizes these energy savings. Due to reduction into these overheads average energy consumed per

instruction drops exponentially. Maximum code base needs to be converted into c- cores to effectively utilize savings provided them. If less amount will be executed on c-cores then energy consumption will increase and is not desired. Figure 3(a) present's energy consumption in conventional baseline CPU, which consumes lots of energy as compared to conservative cores. Energy consumed per instruction will be very large due to various overheads in baseline processor.

G. Other Considerations and Improvements

In recent time some processor has been developed which has removed the problem of dark silicon and utilization wall. In the proposed system conservative cores and accelerator has been used. Software code is synthesized in these c-cores and accelerator and automated tools are required for synthesis. Such automatic tool-chain has been developed which automatically generates placed-and-routed c-core tiles, given the source code and information about execution properties. The cycle and energy accurate simulation tools have confirmed the energy savings provided by c-cores. Currntly the work is being done on more detailed full system emulation to improve the workload modeling so that tools can finalize the selection of c-cores that will populate processors dark silicon. In parallel with this effort, work is being done physical design.

Identification of code for conservative core is very crucial and difficult task. Manual configuration of conservative cores is not possible so, automated tool have been devised to accomplish synthesis. Presently work is being done on development of optimized tools for synthesis. In past accelerator were used for improving efficiency. Both high level synthesis tool and accelerator has been discussed in next subsection.

E. Accelerators

Software code always executes at lower speed then the code executed by the hardware directly. Accelerators has been used in past to boost up performance of execution by converting some codes into hardware. Accelerators can be customized by the programmer for specific use. Here accelerators are actually hardware that offers very fast computation. Parts of the algorithm that can be parallelizable are usually converted in to accelerative hardware. In addition, the codes which are large and are used frequently are converted into accelerators to gain performance efficiency.

One extreme example of such an accelerator is Anton processor, which attains 00 times or more improvement in molecular-dynamics simulation versus general-purpose supercomputer machines. Many functions like baseband processing, 3D graphics, or video decoding or encoding are usually converted in to accelerators. The challenge in creating accelerators is n reorganizing the algorithm to achieve parallel execution. Being able to do this effectively depends on the availability of exploitable parallelism in the algorithm and the ability to expose this parallelism in the form of an accelerator circuit without errors or excessive effort, complexity, or cost.

F. High-level Synthesis Tools

High level synthesis (HLS) tools are used to create accelerators that improve the execution performance of critical algorithms by implementing these algorithms in integrated circuits. HLS research has a long and rich history, which has resulted in the availability of several commercial tools, including AutoESL's AutoPilot, Cadence's C-to-Silicon Compiler.

Because these tools seek to infer parallel execution from serial code, they have many of the same limitations that parallelizing compilers suffer from—namely, the difficulties of analyzing pointers in free form code, extracting memory parallelism, and extracting and formulating efficient parallel schedules for the operations in critical loops. These are difficult tasks that are generally NP-complete or worse in complexity and such difficult code sections are addressed in special way by the automated tools used for synthesis.

III. 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] Taneja, Kavita, Harmunish Taneja, and Rohit Kumar. "SPF: Segmented processor framework for energy efficient proactive routing based applications in MANET." 205 2nd International Conference on Recent Advances in Engineering & Computational Sciences (RAECS). IEEE, 205.

[2] K. Taneja et al., "Cross-Platform Application Development for Smartphones: Approaches and Implications", 3rd International Conference on Computing for Sustainable Global Development, IEEE, 206.
[3] R. K. et. al., "Specialized hardware Architecture for Smartphones", Intl. Journal of Engineering Research and Applications, ISSN: 2248-9622, pp.76-80.
[4] R. K. et al., "Smartphones hardware Architectures and Their Issues", International Journal of Engineering Research and Applications, ISSN: 2248-9622, pp.76-80.
[5] Deepinder Kaur. et al., "Energy Named Entity Recognition, Extraction and Classification using Conditional Random Field with Kernel Approach", IJEAR, ISSN: 0973-4562,Vol. 0, Issue:5, pp. 3893-3898.
[6] S.K. et al., Comparing Routing Protocols, IJEMER, ISSN: 2249-6645, Vol. 4, Issue: 2, pp. 36-42.
[7] K. Taneja et. al., "EEGRP: Grid Based and Caching Equipped Energy Efficient Routing Protocol for MANET with Restricted Movement", Far East Journal of Electronics and Communications, ISSN: 0973-6999 Special Volume 3, Part I, pp. 85-99, 206.
[8] R.K. et. al., "Smartphone's hardware and software development Issues", IJES, ISSN: 2320-0332, pp. 69-75.
[9] Wen-Hwa Liao et al., "GRID: A fully Location Aware Routing Protocols for MANET", Telecommunications Systems, pp. 548-557 (200).
[10] Y.-C. Tseng et al., "Power Saving Protocol for IEEE 802. based Multi-hop MANET", IEEE, pp. 37-60, (200).
[11] Shiv K. Verma et al., "Hybrid Image Fusion Algorithm using Laplace pyramid and PCA Method" Proceedings of ICTS-206 ACM Conference, ISBN 978--4503-3962-9, pp. -6, 206.
[12] L. Pawar et. al., "Optimized Route Selection On The Basis Of Discontinuity And Energy Consumption In Delay Tolerant Networks", Springer: Advances in Intelligent Systems and Computing, ISSN: 294-5357, 206.
[13] R. K. Bhullar et al. "Stress Calculation in Parallel Processor Systems Using Buddy Approach with Enhanced Short Term CPU Scheduling", Proceedings of the International Conference on Communication and Computing Systems (ICCCS 206), ISBN: 978 3802952, 206.
[14] M. Kaur et al., "Stochastic Approach for Energy-Efficient Clustering in WSN", Global Journal of Computer Science and Technology, Vol. 4, Issue 7, pp. -8.
[15] Rohit K. Bhullar et al., "Intelligent Stress Calculation in Segmented Processor Systems Using Buddy Approach", Journal of intelligent and fuzzy systems" IOS press, ISSN: 064-246, In Press.
[16] Rohit K. Bhullar et al, "Test Your Programming Quotient for C", Param-Hans Publisher New Delhi, ISBN: 978--329-80946-8, Book.
[17] Rohit Kumar et. al, "Pen Drive With OS Controlled Inbuilt Permanent Data Storage Partition", Patent 2 /DEL/205.
[18] Rohit Kumar et. al, "Real Time Cleanliness Status Monitoring System For Public Toilets", Indian Patent 957/DEL/205.
[19] R.K. Bhullar et al., "A Novel Prime Numbers Based Hashing Technique for Minimizing Collisions" 206 2nd International Conference on Next Generation Computing Technologies (NGCT-206). IEEE, 206