Evaluated Design of High-Performance Processing Architectures

Utkarsh Rastogi

Department of Computer Engineering and Applications, GLA University, India

Abstract: ***I present the design and evaluation of two new processing elements for reconfigurable computing. I also present a circuit-level implementation of the data paths in static and dynamic design styles to explore the various performance-power tradeoffs involved. When implemented in IBM 90-nm CMOS process, the 8-b data paths achieve operating frequencies ranging over 1 GHz both for static and dynamic implementations, with each data path supporting single- cycle computational capability. A novel single-precision floating point processing element (FPPE) using a 24-b variant of the proposed data paths is also presented. The full dynamic implementation of the FPPE shows that it operates at a frequency of 1 GHz with 6.5-mW average power consumption. Comparison with competing architectures shows that the FPPE provides two orders of magnitude higher throughput. Furthermore, to evaluate its feasibility as a soft- processing solution, we also map the floating point unit onto the Virtex 4 and 5 devices, and observe that the unit requires less than 1% of the total logic slices, while utilising only around 4% of the DSP blocks available. When compared against popular field-programmable-gate-array-based floating point units, our design on Virtex 5 showed significantly lower resource utilisation, while achieving comparable peak operating frequency. 3D integration of solid-state memories and logic, as demonstrated by the Hybrid Memory Cube (HMC), offers major opportunities for revisiting near-memory computation and gives new hope to mitigate the power and performance losses caused by the "memory wall". Several publications in the past few years demonstrate this renewed interest. In this paper we present the first exploration steps towards design of the Smart Memory Cube (SMC), a new Processor-in- Memory (PIM) architecture that enhances the capabilities of the logic-base (LoB) die in HMC. An accurate simulation environment called SMCSim has been developed, along with a full featured software stack. The key contribution of this work is full system analysis of near memory computation including high-level software to low-level firmware and hardware layers, considering offloading and dynamic overheads caused by the operating system (OS), cache coherence, and memory management. A zero-copy pointer passing mechanism has been devised to allow low overhead data sharing between the host and the PIM. Benchmarking results demonstrate up to 2X performance improvement in comparison with the host System-on-Chip (SoC), and around 1.5X against a similar host-side accelerator. Moreover, by scaling down the voltage and frequency of PIM's processor it is possible to reduce energy by around 70 % and 55 % in comparison with the host and the accelerator, respectively.***

Keywords: High Performance Computing, HiPC, Low End Computing, Super Computers, Personal Computers

1. Introduction

At a hardware level, the overall system performance of these architectures depends on: 1) the top-level array and interconnection scheme and 2) the individual processing cell. While factors such as organisation of the cells, interconnection network, and memory hierarchy (in case of shared memory architectures) are critical to system throughput, it should be noted that the individual processing cells are the main workhorses of the system and hence are perhaps equally critical to the total processing throughput. It is therefore important to develop extendable arithmetic processing units which allow modular system design in order to guarantee maximum throughput from a reconfigurable array-based architecture. Another important requirement of modern DSP and media processing applications is to provide floating-point capability. This ability, if achieved by reusing or extending integer data paths, allows faster development time, low-cost system implementation, as well as possible FPGA implementation of the data paths.

In this paper, I present the architectures of two integer reconfigurable data paths. The proposed data paths can perform single-cycle addition, subtraction, multiplication, and accumulation operations. They can be used in multicore platforms to perform more complex arithmetic and logical operations. The data paths have a short and uniform critical path across the range of operations. Each of the data paths is extendable and can be parameterised to support higher precision arithmetic, and software-assisted variable-precision reconfigurable systems. Eight-bit versions of the integer data paths were implemented using the IBM 90-nm process using static, domino, and data-driven dynamic logic (D3L). Simulation results show that the data paths can achieve operating frequencies in the range of 1 GHz. Using the findings from the architectural and circuit analysis on the integer data paths, a new single-precision floating point processing element (FPPE) using the 24-b extension of the data paths is also presented. The full dynamic implementation of the FPPE operates at a frequency of 1 GHz with 6.5-mW average power consumption.

To understand the feasibility of the proposed data paths for FPGA applications, we also performed synthesis experiments using Xilinx Virtex 4 and 5 FPGAs. These experiments helped to understand the tradeoffs associated with choosing optimum granularities and the impact of modularising large-width operations on system throughput. The FPPE was also synthesized to evaluate its potential as a soft floating point PE. Comparative analysis with competing architectures shows that the proposed FPPE achieves comparable performance at significantly lower resource utilisation.

Multiprocessing System-on-chip
Multiprocessor system-on-chip (MP-SoC) platforms are emerging as an important trend for SoC design. Power and wire design constraints are forcing the adoption of new design methodologies for system-on-chip (SoC), namely, those that incorporate modularity and explicit parallelism.

To enable these MP-SoC platforms, researchers have recently pursued scalable communication-centric interconnect fabrics, such as networks-on-chip (NoC), which possess many features that are particularly attractive for these. These communication-centric interconnect fabrics are characterised by different trade-offs with regard to latency, throughput, energy dissipation, and silicon area requirements. In this paper, we develop a consistent and meaningful evaluation methodology to compare the performance and characteristics of a variety of NoC architectures. We also explore design trade-offs that characterise the NoC approach and obtain comparative results for a number of common NoC topologies. To the best of our knowledge, this is the first effort in characterising different NoC architectures with respect to their performance and design trade-offs. To further illustrate our evaluation methodology, we map a typical multiprocessing platform to different NoC interconnect architectures and show how the system performance is affected by these design trade-offs.

2. Processor Design

Processor design is the design engineering task of creating a processor, a key component of computer hardware. It is a subfield of computer engineering (design, development and implementation) and electronics engineering (fabrication). The design process involves choosing an instruction set and a certain execution paradigm (e.g. VLIW or RISC) and results in a microarchitecture, which might be described in e.g. VHDL or Verilog. For microprocessor design, this description is then manufactured employing some of the various semiconductor device fabrication processes, resulting in a die which is bonded onto a chip carrier. This chip carrier is then soldered onto, or inserted into a socket on, a printed circuit board (PCB). The mode of operation of any processor is the execution of lists of instructions. Instructions typically include those to compute or manipulate data values using registers, change or retrieve values in read/write memory, perform relational tests between data values and to control program flow.

Basics-

CPU design is divided into design of the following components:

- Data paths (such as ALUs and pipelines)
- Control unit: logic which controls the data paths
- Memory components such as register files, caches
- Clock circuitry such as clock drivers, PLLs, clock distribution networks
- Pad transceiver circuitry
- Logic gate cell library which is used to implement the logic

CPUs designed for high-performance markets might require custom (optimised or application specific (see below)) designs for each of these items to achieve frequency, power-dissipation, and chip-area goals whereas CPUs designed for lower performance markets might lessen the implementation burden by acquiring some of these items by purchasing them as intellectual property. Control logic implementation techniques (logic synthesis using CAD tools) can be used to implement data paths, register files, and clocks. Common logic styles used in CPU design include unstructured random logic, finite-state machines, microprogramming (common from 1965 to 1985), and Programmable logic arrays (common in the 1980s, no longer common).

Implementation logic

Device types used to implement the logic include:

- Transistor-transistor logic Small Scale Integration logic chips - no longer used for CPUs
- Programmable Array Logic and Programmable logic devices - no longer used for CPUs
- Emitter-coupled logic (ECL) gate arrays - no longer common
- CMOS gate arrays - no longer used for CPUs
- CMOS mass-produced ICs - the vast majority of CPUs by volume
- CMOS ASICs - only for a minority of special applications due to expense
- Field-programmable gate arrays (FPGA) - common for soft microprocessors, and more or less required for reconfigurable computing

A CPU design project generally has these major tasks:

- Programmer-visible instruction set architecture, which can be implemented by a variety of microarchitectures
- Architectural study and performance modelling in ANSI C/C++ or SystemC
- High-level synthesis (HLS) or register transfer level (RTL, e.g. logic) implementation
- RTL verification
- Circuit design of speed critical components (caches, registers, ALUs)
- Logic synthesis or logic-gate-level design
- Timing analysis to confirm that all logic and circuits will run at the specified operating frequency
- Physical design including floor planning, place and route of logic gates
- Checking that RTL, gate-level, transistor-level and physical-level representations are equivalent
- Checks for signal integrity, chip manufacturability

Re-designing a CPU core to a smaller die-area helps to shrink everything (a "photomask shrink"), resulting in the same number of transistors on a smaller die. It improves performance (smaller transistors switch faster), reduces power (smaller wires have less parasitic capacitance) and reduces cost (more CPUs fit on the same wafer of silicon). Releasing a CPU on the same size die, but with a smaller CPU core, keeps the cost about the same but allows higher levels of integration within one very-large-scale integration chip (additional cache, multiple CPUs or other components), improving performance and reducing overall system cost.

As with most complex electronic designs, the logic verification effort (proving that the design does not have bugs) now dominates the project schedule of a CPU.

Key CPU architectural innovations include index register, cache, virtual memory, instruction pipelining, superscalar, CISC, RISC, virtual machine, emulators, microprogram, and stack.

Performance analysis and benchmarking

Benchmarking is a way of testing CPU speed. Examples include SPECint and SPECfp, developed by Standard Performance Evaluation Corporation, and ConsumerMark developed by the Embedded Microprocessor Benchmark Consortium EEMBC.

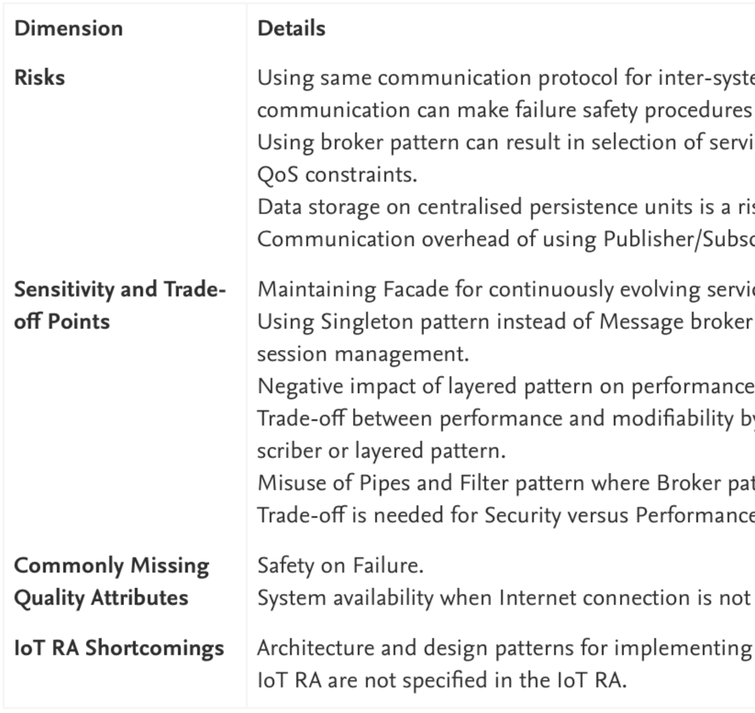
Some of the commonly used metrics include:

- Instructions per second - Most consumers pick a computer architecture (normally Intel IA32 architecture) to be able to run a large base of pre-existing pre-compiled software. Being relatively uninformed on computer benchmarks, some of them pick a particular CPU based on operating frequency (see Megahertz Myth).
- FLOPS - The number of floating point operations per second is often important in selecting computers for scientific computations.
- Performance per watt - System designers building parallel computers, such as Google, pick CPUs based on their speed per watt of power, because the cost of powering the CPU outweighs the cost of the CPU itself.
- Some system designers building parallel computers pick CPUs based on the speed per dollar.
- System designers building real-time computing systems want to guarantee worst-case response. That is easier to do when the CPU has low interrupt latency and when it has deterministic response. (DSP)
- Computer programmers who program directly in assembly language want a CPU to support a full featured instruction set.
- Low power - For systems with limited power sources (e.g. solar, batteries, human power).
- Small size or low weight - for portable embedded systems, systems for spacecraft.
- Environmental impact - Minimising environmental impact of computers during manufacturing and recycling as well during use. Reducing waste, reducing hazardous materials. (search Green computing).

There may be tradeoffs in optimising some of these metrics. In particular, many design techniques that make a CPU run faster make the "performance per watt", "performance per dollar", and "deterministic response" much worse, and vice versa.

Architecture Evaluation Activities

Architecture Evaluation activities consisted of three stages. (i) Before the evaluation session, the groups prepared a short architecture evaluation questionnaire on quality attributes considered in the architecture design, key architecture design decisions, strengths and weaknesses of the design decisions, sensitivity and trade-off points, and risks and non-risks in the architecture [26]. (ii) During the evaluation sessions, the group whose architecture was evaluated presented the architecture while the group who was evaluating the architecture asked questions based upon their initial preparation as well as from integration perspective of their own architecture. During the architecture presentation, design artefacts were analysed for evaluation of the IoT subsystem architectures as well some new artefacts specific to architecture evaluation such as architecture utility trees [21] were generated to present architecture design decision corresponding to quality attributes. During the architecture evaluation sessions, sensitivity points, trade off points, architecture risks and architecture non risks were discussed. (iii) After the individual architecture evaluation sessions, a joint session was conducted in which each group briefly presented their IoT subsystem architecture, quality attributed those were considered in the architecture and feedback it received during the individual architecture evaluation session. Each group member also prepared a written report on the evaluation of the architecture that the group member had evaluated in the architecture evaluation session. The report was shared with teaching staff, who was responsible for design and analysis of the WoT architecture. The following shows commonly reported missing quality requirements, risky design decisions, sensitivity and trade-off points, and common improvements suggested in the IoT subsystem architectures during the evaluation sessions.



Dimension	Details
Risks	Using same communication protocol for inter-syste communication can make failure safety procedures Using broker pattern can result in selection of servi QoS constraints. Data storage on centralised persistence units is a ris Communication overhead of using Publisher/Subsc
Sensitivity and Trade-off Points	Maintaining Facade for continuously evolving servi Using Singleton pattern instead of Message broker session management. Negative impact of layered pattern on performance Trade-off between performance and modifiability by scriber or layered pattern. Misuse of Pipes and Filter pattern where Broker pat Trade-off is needed for Security versus Performance
Commonly Missing Quality Attributes	Safety on Failure. System availability when Internet connection is not
IoT RA Shortcomings	Architecture and design patterns for implementing IoT RA are not specified in the IoT RA.

3. Methods

Performance evaluation is at the foundation of computer architecture research and development. Contemporary microprocessors are so complex that architects cannot design systems based on intuition and simple models only. Adequate performance evaluation methods are absolutely crucial to steer the research and development process in the right direction. However, rigorous performance evaluation is non-trivial as there are multiple aspects to performance evaluation, such as picking workloads, selecting an appropriate modelling or simulation approach, running the model and interpreting the results using meaningful metrics. Each of these aspects is equally important and a performance evaluation method that lacks rigour in any of these crucial aspects may lead to inaccurate performance data and may drive research and development in a wrong direction. The goal of this book is to present an overview of the current state-of-the-art in computer architecture performance evaluation, with a special emphasis on methods for exploring processor architectures. The book focuses on fundamental concepts and ideas for obtaining accurate performance data. The book covers various topics in performance evaluation, ranging from performance metrics, to workload selection, to various modelling approaches including mechanistic and empirical modelling. And because simulation is by far the most prevalent modelling technique, more than half the book's content is devoted to simulation. The book provides an overview of the simulation techniques in the computer designer's toolbox, followed by various simulation acceleration techniques including sampled simulation, statistical simulation, parallel simulation and hardware-accelerated simulation.

4. Analysing Architectures

Our tour of the ABC has gotten us to the stage where an architect has designed and documented an architecture. This leads us to discuss how to evaluate or to analyse the architecture to make sure it is the one that will do the job. That is the focus of which we begin by answering some basic questions about architectural evaluations-why, when, cost, benefits, techniques, planned or unplanned, preconditions, and results.

Why
One of the most important truths about the architecture of a system is that knowing it will tell you important properties of the system itself-even if the system does not yet exist. Architects make design decisions because of the downstream effects they will have on the system(s) they are building, and these effects are known and predictable. If they were not, the process of crafting an architecture would be no better than throwing dice: We would pick an architecture at random, build a system from it, see if the system had the desired properties, and go back to the drawing board if not. While architecture is not yet a cookbook science, we know we can do much better than random guessing.

Architects by and large know the effects their design decisions will have. As we saw in architectural tactics and patterns in particular bring known properties to the systems in which they are used. Hence, design choices-that is to say, architectures-are analysable. Given an architecture, we can deduce things about the system, even if it has not been built yet.

Why evaluate an architecture? Because so much is riding on it, and because you can. An effective technique to assess a candidate architecture-before it becomes the project's accepted blueprint-is of great economic value. With the advent of repeatable, structured methods (such as the ATAM), architecture evaluation has come to provide relatively a low-cost risk mitigation capability. Making sure the architecture is the right one simply makes good sense. An architecture evaluation should be a standard part of every architecture-based development methodology.

When
It is almost always cost-effective to evaluate software quality as early as possible in the life cycle. If problems are found early, they are easier to correct-a change to a requirement, specification, or design is all that is necessary. Software quality cannot be appended late in a project, but must be inherent from the beginning, built in by design. It is in the project's best interest for prospective candidate designs to be evaluated (and rejected, if necessary) during the design phase, before long-term institutionalisation.

However, architecture evaluation can be carried out at many points during a system's life cycle. If the architecture is still embryonic, you can evaluate those decisions that have already been made or are being considered. You can choose among architectural alternatives. If the architecture is finished, or nearly so, you can validate it before the project commits too lengthy and expensive development. It also makes sense to evaluate the architecture of a legacy system that is undergoing modification, porting, integration with other systems, or other significant upgrades. Finally, architecture evaluation makes an excellent discovery vehicle: Development projects often need to understand how an inherited system meets (or whether it meets) its quality attribute requirements.

Furthermore, when acquiring a large software system that will have a long lifetime, it is important that the acquiring organisation develop an understanding of the underlying architecture of the candidate. This makes an assessment of their suitability possible with respect to qualities of importance.

Evaluation can also be used to choose between two competing architectures by evaluating both and seeing which one fares better against the criteria for "goodness."

Cost
The cost of an evaluation is the staff time required of the participants. AT&T, having performed approximately 300 full-scale architecture reviews on projects requiring a minimum of 700 staff-days, reported that, based on estimates from individual project managers, the average cost was 70 staff-days. ATAM-based reviews require approximately 36 staff-days.If your organisation adopts a standing unit for carrying out evaluations, then costs for supporting it must be included, as well as time to train the members. [1] These figures are for the evaluation team. The ATAM also requires participation from project stakeholders and decision makers, which adds to the total.

Benefits

We enumerate six benefits that flow from holding architectural inspections.

1) Financial. At AT&T, each project manager reports perceived savings from an architecture evaluation. On average, over an eight-year period, projects receiving a full architecture evaluation have reported a 10% reduction in project costs. Given the cost estimate of 70 staff-days, this illustrates that on projects of 700 staff-days or longer the review pays for itself. Other organisations have not publicised such strongly

quantified data, but several consultants have reported that more than 80% of their work was repeat business. Their customers recognised sufficient value to be willing to pay for additional evaluations. There are many anecdotes about estimated cost savings for customers' evaluations. A large company avoided a multi-million-dollar purchase when the architecture of the global information system they were procuring was found to be incapable of providing the desired system attributes. Early architectural analysis of an electronic funds transfer system showed a $50 billion transfer capability per night, which was only half of the desired capacity. An evaluation of a retail merchandise system revealed early that there would be peak order performance problems that no amount of hardware could fix, and a major business failure was prevented. And so on. There are also anecdotes of architecture evaluations that did not occur but should have. In one, a rewrite of a customer accounting system was estimated to take two years but after seven years the system had been reimplemented three times. Performance goals were never met despite the fact that the latest version used sixty times the CPU power of the original prototype version. In another case, involving a large engineering relational database system, performance problems were largely attributable to design decisions that made integration testing impossible. The project was canceled after $20 million had been spent.

2) Forced preparation for the review. Indicating to the reviews the focus of the architecture evaluation and requiring a representation of the architecture before the evaluation is done means that reviews must document the system's architecture. Many systems do not have an architecture that is understandable to all developers. The existing description is either too brief or (more commonly) too long, perhaps thousands of pages. Furthermore, there are often misunderstandings among developers about some of the assumptions for their elements. The process of preparing for the evaluation will reveal many of these problems.
3) Captured rationale. Architecture evaluation focuses on a few specific areas with specific questions to be answered. Answering these questions usually involves explaining the design choices and their rationales. A documented design rationale is important later in the life cycle so that the implications of modifications can be assessed. Capturing a rationale after the fact is one of the more difficult tasks in software development. Capturing it as presented in the architecture evaluation makes invaluable information available for later use.
4) Early detection of problems with the existing architecture. The earlier in the life cycle that problems are detected, the cheaper it is to fix them. The problems that can be found by an architectural evaluation include unreasonable (or expensive) requirements, performance problems, and problems associated with potential downstream modifications. An architecture evaluation that exercises system modification scenarios can, for example, reveal portability and extensibility problems. In this way an architecture evaluation provides early insight into product capabilities and limitations.
5) Validation of requirements. Discussion and examination of how well an architecture meets requirements opens up the requirements for discussion. What results is a much clearer understanding of the requirements and, usually, their prioritisation. Requirements creation, when isolated from early design, usually results in conflicting system properties. High performance, security, fault tolerance, and low cost are all easy to demand but difficult to achieve, and often impossible to achieve simultaneously. Architecture evaluations uncover the conflicts and tradeoffs, and provide a forum for their negotiated resolution.
6) Improved architectures. Organisations that practice architecture evaluation as a standard part of their development process report an improvement in the quality of the architectures that are evaluated. As development organisations learn to anticipate the questions that will be asked, the issues that will be raised, and the documentation that will be required for evaluations, they naturally pre-position themselves to maximise their performance on the evaluation. Architecture evaluations result in better architectures not only after the fact but before the fact as well. Over time, an organisations develops a culture that promotes good architectural design.

In sum, architecture evaluations tend to increase quality, control cost, and decrease budget risk. Architecture is the framework for all technical decisions and as such has a tremendous impact on product cost and quality. An architecture evaluation does not guarantee high quality or low cost, but it can point out areas of risk. Other factors, such as testing or quality of documentation and coding, contribute to the eventual cost and quality of the system.

Techniques

The ATAM and CBAM methods discussed in the next two chapters are examples of questioning techniques. Both use scenarios as the vehicle for asking probing questions about how the architecture under review responds to various situations. Other questioning techniques include checklists or questionnaires. These are effective when an evaluation unit encounters the same kind of system again and again, and the same kind of probing is appropriate each time. All questioning techniques essentially rely on thought experiments to find out how well the architecture is suited to its task.

Complementing questioning techniques are measuring techniques, which rely on quantitative measures of some sort. One example of this technique is architectural metrics. Measuring an architecture's coupling, the cohesiveness of its modules, or the depth of its inheritance hierarchy suggests something about the modifiability of the resulting system. Likewise, building simulations or prototypes and then measuring them for qualities of interest (here, runtime qualities such as performance or availability) are measuring techniques.

While the answers that measuring techniques give are in some sense more concrete than those questioning techniques provide, they have the drawback that they can be applied only in the presence of a working artefact. That is, measuring techniques have to have something that exists that can be measured. Questioning techniques, on the other hand, work just fine on hypothetical architectures, and can be applied much earlier in the life cycle.

Planned or Unplanned

Evaluations can be planned or unplanned. A planned evaluation is considered a normal part of the project's development cycle. It is scheduled well in advance, built into the project's work plans and budget, and follow-up is expected. An unplanned evaluation is unexpected and usually the result of a project in serious trouble and taking extreme measures to try to salvage previous effort.

The planned evaluation is ideally considered an asset to the project, at worst a distraction from it. It can be perceived not as a challenge to the technical authority of the project's members but as a validation of the project's initial direction. Planned evaluations are pro-active and team-building.

An unplanned evaluation is more of an ordeal for project members, consuming extra project resources and time in the schedule from a project already struggling with both. It is initiated only when management perceives that a project has a substantial possibility of failure and needs to make a mid-course correction. Unplanned evaluations are reactive, and tend to be tension filled. An evaluation's team leader must take care not to let the activities devolve into finger pointing. Needless to say, planned evaluations are preferable.

5. Preconditions

A successful evaluation will have the following properties:

1) Clearly articulated goals and requirements for the architecture. An architecture is only suitable, or not, in the presence of specific quality attributes. One that delivers breathtaking performance may be totally wrong for an application that needs modifiability. Analysing an architecture without knowing the exact criteria for "goodness" is like beginning a trip without a destination in mind. Sometimes (but in our experience, almost never), the criteria are established in a requirements specification. More likely, they are elicited as a precursor to or as part of the actual evaluation. Goals define the purpose of the evaluation and should be made an explicit portion of the evaluation contract, discussed subsequently.
2) Controlled scope. In order to focus the evaluation, a small number of explicit goals should be enumerated. The number should be kept to a minimum-around three to five-an inability to define a small number of high-priority goals is an indication that the expectations for the evaluation (and perhaps the system) may be unrealistic.
3) Cost-effectiveness. Evaluation sponsors should make sure that the benefits of the evaluation are likely to exceed the cost. The types of evaluation we describe are suitable for medium and large-scale projects but may not be cost-effective for small projects.
4) Key personnel availability. It is imperative to secure the time of the architect or at least someone who can speak authoritatively about the system's architecture and design. This person (or these people) primarily should be able to communicate the facts of the architecture quickly and clearly as well as the motivation behind the architectural decisions. For very large systems, the designers for each major component need to be involved to ensure that the architect's notion of the system design is in fact reflected and manifested in its more detailed levels. These designers will also be able to speak to the behavioural and quality attributes of the components. For the ATAM, the architecture's stakeholders need to be identified and represented at the evaluation. It is essential to identify the customer(s) for the evaluation report and to elicit their values and expectations.
5) Competent evaluation team. Ideally, software architecture evaluation teams are separate entities within a corporation, and must be perceived as impartial, objective, and respected. The team must be seen as being composed of people appropriate to carry out the evaluation, so that the project personnel will not regard the evaluation as a waste of time and so that its conclusions will carry weight. It must include people fluent in architecture and architectural issues and be led by someone with solid experience in designing and evaluating projects at the architectural level.
6) Managed expectations. Critical to the evaluation's success is a clear, mutual understanding of the expectations of the organization sponsoring it. The evaluation should be clear about what its goals are, what it will produce, what areas it will (and will not) investigate, how much time and resources it will take from the project, and to whom the results will be delivered.

6. Acknowledgement

I would like to say Thanks to Centra, Universidade Téchnia de Lisboa, Sergio Almeida, Lisbon Gravity Group, Professor Vitor Cardoso for letting me access the relevant data and information for my research. I further want to thank my mentor Dr. Manoj Choubey who had introduced me to High Performance computing and also helped me on parallel/concurrent computing that really helped to clarify the subject.

References

[1] https://www.sciencedirect.com/science/article/pii/S1389128602004541
[2] https://ieeexplore.ieee.org/abstract/document/1453503
[3] https://www.sciencedirect.com/topics/computer-science/architecture-evaluation
[4] http://www.ece.ubc.ca/~matei/EECE417/BASS/part03.html
[5] ”EEMBC ConsumerMark". Archived from the original on March 27, 2005.
[6] Stephen Shankland (December 9, 2005). "Power could cost more than servers, Google warns”.
[7] Kerr, Justin. "AMD Loses Market Share as Mobile CPU Sales Outsell Desktop for the First Time." Maximum PC. Published 2010-10-26.
[8] "New system manages hundreds of transactions per second" article by Robert Horst and Sandra Metz, of Tandem Computers Inc., "Electronics" magazine, 1984 April 19: "While most high-performance CPUs require four to five years to develop, The NonStopTXP

processor took just 2+1/2 years -- six months to develop a complete written specification, one year to construct a working prototype, and another year to reach volume production.”

[9] S. Mittal, "A survey of techniques for improving energy efficiency in embedded computing systems", IJCAET, 6(4), 440–459, 2014.

[10] Curtis A. Nelson. 8051 overview. Archived from the original pdf on 2011-10-09. Retrieved 2011-07-10

[11] Square millimetres per 8051, 0.013 in 45nm line-widths;

[12] To figure dollars per square millimetres, see [1], and note that an SOC component has no pin or packaging costs.

[13] "ARM Cores Climb Into 3G Territory" by Mark Hachman, 2002.

[14] "The Two Percent Solution" by Jim Turley 2002.

[15] "ARM's way" 1998

[16] "Why the Propeller Works" by Chip Gracey

[17] "Interview with William Mensch"

[18] C.H. Séquin; D.A. Patterson. "Design and Implementation of RISC I" (PDF).

[19] "the VHS". Archived from the original on 2010-02-27.

[20] Jan Gray. "Teaching Computer Design with FPGAs".

[21] Norman P. Jouppi; Jeffrey Y. F. Tang (1989). "A 20-MIPS Sustained 32-bit CMOS Microprocessor with High Ratio of Sustained to Peak Performance". p. i. CiteSeerX 10.1.1.85.988

[22] "MultiTitan: Four Architecture Papers" (PDF). 1988. pp. 4–5.