Profiles in Power: Optimizing Real-Time Systems for Power

Graham R. Hellestrand
g.hellestrand@vastsystems.com

Mahdi Seddighnezhad
m.seddighnezhad@vastsystems.com

James E. Brogan
j.brogan@vastsystems.com

VaST Systems Technology Corporation
1250 Oakmead Pkwy, Suite 310
Sunnyvale, CA 94085
+1-408-328-0949

ABSTRACT

High-performance, timing accurate models of complex systems (called Virtual System Prototypes (VSP)) enable the computation of relatively accurate power in terms of events that occur in the model. VSPs are the integrations of models of electronic hardware, communication and mechanical subsystems into systems that execute software accurately. Software has a first order impact on system performance and has, typically, the major effect on modern system optimization. The computation of relative power, although fundamental, is not useful by itself – doubling the *talk* time of a mobile phone is not useful if, concomitantly, the speed dwindles so that look-up functions take 20 seconds rather than the 2 seconds that competitors take. Power is an exemplar of the complex, *concept*-based functions, with many hardware, software and system parameters, that constitute optimization functions and will be treated in detail in this paper. The general form of a power computation function is given in the paper, as well as, a simple example of the implementation of a power calculator. The use of power, along with the other components of objective functions, such as speed (instructions per second), response latency and cost, must drive algorithm choice and software development in mobility and other power-performance sensitive applications. The use of VSPs is mandatory in specifying the hardware and software architectures of, and then building, complex optimal systems.

Categories and Subject Descriptors

C3 **[Special Purpose and Application Based Systems]** Real-time and embedded system; C4 **[Performance of Systems]** Measurement and modeling techniques; G3 **[Probability and Statistics]** Experimental design.

General Terms

Design, Experimentation, Measurement, Performance.

Keywords

Power measurement and analysis, quantitative systems architecture, empirical system design, event-based objective function, event data-driven optimization.

1. Background and Motivation

An empirical approach to composing optimal architectures for application specific embedded systems is relatively rare. The use of empiricism in developing optimal software is even rarer, and when used often primitive. The complexity of processor centric, electronic systems that control modern products (such as, cell phones, automobiles, communication base stations, consumer products) requires a systematic approach to developing software in order to deliver an optimal fit for an intended product. When a company's engineering process is being used as a competitive weapon, the luxury of optimality, especially wrt power and speed and response latency in mobile systems, becomes a necessity [1].

The bigger architecture picture is more complex. The intuitive optimization of systems – architecture, software design, hardware design, and interfaces – has largely been a by-product of hardware design. Since hardware designers have rarely understood, or had access to, the software that would run on their architectures, they produced conservative, often grossly over-engineered designs that were typically poor fits to a number of dimensions of the specification - especially in cost sensitive applications, where over-engineering is the antithesis of cost sensitivity [2].

The ability to support data-driven decision making early in the software development process is one of the underlying drivers of building models of systems that are timing accurate and high performance. Of course, this advantage also accrues to architecture development but that is another dialogue. From a purely software perspective, optimizing across the dimension of speed, response latency, cost (size) and power consumption is rarely done and, at a pre-silicon level, it is an undertaking only possible using high-performance, timing accurate models – called VSPs in this paper.

It is known that poor software and inefficient algorithms have a 1st order effect on an embedded system's performance. This is difficult to reconcile with practice, when next-generation product planning has prime foci of processor microarchitecture and hardware (platform) design, regardless of the fact that iterative processor microarchitecture improvement typically yields a 2nd or 3rd order effect and hardware (platform) design has a 1st order effect at the system level. The question is what happened to software? It is negligent not to employ empirical methods in the development of software in real-time, embedded systems.

For complex, multi-processor and networked real-time systems running full software loads (i) *advances in computer architecture and software have made it difficult or impossible to estimate or predict software's execution time* [8] and (ii) experience with proving the correctness of even a well constructed, small, single processor micro-kernel, L4, indicates the overwhelming complexity of applying this technology to more complex systems [9]. This leaves schedulability analysis in the difficult position of having to use traces from simulation to determine whether a system will meets its overall real-time constraints. Deriving best-case, average and worst-case system performance from simulation using VSPs enables analyses of system variability leading to the identification of factors having the most significant impact on variability. These factors are prognostic as well as diagnostic and they can be used to drive the structural optimization of systems. On the deficit side, the number of simulation may require many experiments to be performed on various configurations of a system. However, here the design of experiment methodology [5] helps by providing a statistically valid mechanism for dramatically reducing the number of experiments needed to be performed.

Since VSPs are used to directly execute software, including hard real-time code, during development and debugging, trace information (streamed from non-perturbing probes inserted into the model) - including response latencies, power consumption, speed between markers, frequency of function calls, etc. - is produced alongside the usual debug data and hence is available to software and systems engineers as a normal part of the edit-compile-execute-debug software development cycle. This changes the perspective of where optimization should occur in the development cycle, and it is not as a post development exercise.

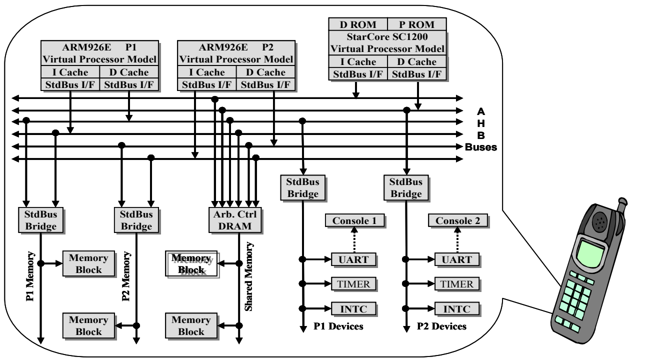


Figure 1: A Typical Virtual System Prototype for Mobility

Figure 1 shows the hardware platform component of a VSP, called a Virtual Prototype, used in the experiments in this paper.

2. Formulating Power

There are many ways of constructing objective functions including for power. The classical way is to track event frequencies and/or latencies and to construct the power function based on events that contribute significantly to the computation of power.

2.1 Event-Based Power Functions

In an event driven simulation environment, a general form of the power function can be expressed as a function whose parameters are functions each characterizing contributions to the objective function by one of the components constituting the system, viz. CPUs, buses, bus bridges, memories and peripheral devices. The parameter functions themselves have parameters that are functions of simulation event types sourced from the various event activities that occur in a VSP during simulation. In general, a power function will have the form shown in Equation 1 [3].

$$
\begin{aligned}
&Equation\,1:| \\
&F_{Power}(|\, f_{CPU}(\Theta_{cc=0..cn} \circ f_{CPU_{cc}}(\Theta_{CEvType=1..cet} \circ g_{CPU_{cc,CEvType}}(\Theta_{CEvCnt=scecn..tcecn} \circ Event_{CPU_{cc,CEvType,CEvCnt}})), \\
&\quad f_{Bus}(\Theta_{bc=0..bcn} \circ f_{Bus_{bc}}(\Theta_{BEvType=1..bet} \circ g_{Bus_{bc,BEvType}}(\Theta_{BEvCnt=sbecn..tbecn} \circ Event_{Bus_{bc,BEvType,BEvCnt}})), \\
&\quad f_{BusBridge}(\Theta_{bbc=0..bbcn} \circ f_{BBus_{bbc}}(\Theta_{BBEvType=1..bbet} \circ g_{BBus_{bbc,BBEvType}}(\Theta_{BBEvCnt=sbbecn..tbbecn} \circ Event_{BBus_{bc,BBEvType,BBEvCnt}})), \\
&\quad f_{Mem}(\Theta_{mc=0..mcn} \circ f_{Mem_{mc}}(\Theta_{MEvType=1..met} \circ g_{Mem_{mc,MEvType}}(\Theta_{MEvCnt=smecn..tmecn} \circ Event_{Mem_{mc,MEvType,MEvCnt}})), \\
&\quad f_{Dev}(\Theta_{dc=0..cn} \circ f_{Dev_{dc}}(\Theta_{DEvType=1..\det} \circ g_{Dev_{dc,DEvType}}(\Theta_{DEvCnt=sdecn..tdecn} \circ Event_{Dev_{dc,DEvType,DEvCnt}})))
\end{aligned}
$$

$$
where\;\; f_{CPU_k}(\Theta_{EvType=1..et} \circ g_{CPUk,EvType}(\ldots)) = f_{CPU_k}(g_{CPU_k,1}(\ldots), g_{CPU_k,2}(\ldots)\;,\ldots, g_{CPU_k,et}(\ldots))
$$

Table 1: Component Event Binding Table

Component Types Binding	**Component Instance Binding**	**Component Event Binding**
f_{CPU}	$f_{ARM1156T2F}$	$f_{SC1200_{TCMWrite}}$ $f_{SC1200_{BranchTaken}}$ …… $f_{SC1200_{LDD}}$
	f_{SC1200}	
	….	

A simple way to visualize and compute a power function is to build an interpretation table, as in Table 1. These tables are large and even though the *Event Bindings* are simple to implement, typically a pointer to a function and a history buffer of events, the extraction of appropriate data from register transfer (RT) models or representative samples of the silicon to put into the tables is not automatic and is difficult and time consuming.

3. Computing Power

We instrumented the VSP of Figure 1 and for the purposes of experiments for this paper put the 2nd ARM926 processor and the Starcore SC1200 processor in *Reset* – so they consumed no cycles and no power.

The basic function computed is Instant Power which calculates the total energy consumed over some period of time or some number of events (such as cycles).

The functions computed that are useful for optimization purposes are:

- Maximum power consumed, over a particular period (maximum of the instant powers)
- Average power consumed over the whole experiment.

A simplified function used to compute instant power per k-cycles is given in the Equation 2:

Similar functions occur for f_{Pipe}, f_{Cache}, f_{TLB}, f_{RegAcc}, f_{MemAcc}, $f_{PeriphAcc}$ and the weights for the constituent *accumulating* functions are given in Table 2, and the weights (W_i) for each of the classes of functions contributing to f_{Power} have been set to the constant function 1 in this study. In more complex studies, the *accumulating* function might be replaced with individual functions relevant to computing power in ways not considered for the simple examples of this paper. Such functions can include history and implementation dependent technology functions. Similarly, the weights (W_i) may be more complex functions – for example, the cache hit weights are functions of cache structure (size, wayness, policies).

4. Experimentation

The following is an outline of the experimental design process.

i. The goal of the 4 experiments reported here was to investigate the effects of various arrangements of cache, buses, memory hierarchy and algorithms on average power consumption and speed. The VSP used is that shown in Figure 1, but with only one ARM926E processor enabled. The target codes selected were MontaVista Linux v2.6, Viterbi and Sieve programs from the EEMBC [4] test suite, and a prime number program downloaded from the web [5]. Access to customer data was not possible for this study.

ii. To determine the goal, we specified, across the executed target codes:
  - Power in terms of average power per instruction executed;
  - Speed in terms of instructions executed per k-cycle (IPC_k);
  - Cost – where cache size was used as a direct indicator of cost

The contributing factors (independent of target codes) to the computation of power were identified as events captured from the VSP. These events are delineated above in Equation 2 and Table 2. The computation of speed is a simpler function – the total number of instructions executed averaged across all cycles executed. This information is directly available from the simulation.

Table 2: Power: Function Types, Event & Weighting Functions

Function Types	Events	Weight Functions
Pipeline	ibase	6.0
Instruction Types	ijmp	2.0
	iexcept	2.0
	ictrl	0
	icoproc	12.0
	iundefs	0
	imemrd	0
	imemwt	0
	imemrw	0
	iarith	1.0
	iother	1.0
Caches (I&D)	Cache_lookup	$f_{i\text{-}dcache}$(size, ways)
	icache_hit	iCache-lookup + f_{icache}(line size, decode)
	icache_miss	Icache_lookup
	dcache_hit	Dcache_lookup + f_{dcache}(size, ways, line size,)
	dcache_miss	Dcache_lookup
	line_fill	0
TLB	tlb_miss	30.0
Register	regfile_access	1.0
Memory (incl. bus transactions)	membus_transaction	50.0
Periph Device (incl. bus transactions)	periphbus_reg_access	50.0

iii. In a simulation environment, all factors are effectively controllable. Therefore randomization of experiments will have no effect. However, sample size and selection – say the random selection of a number of the EEMBC [4] communications related programs – are indeed important parts of the experimental protocol. It is in this way that

Equation 2 :

$$f_{Power} = .W_{Pipe} \times f_{Pipe} + W_{Instr} \times f_{Instr} + W_{Cache} \times f_{Cache} + W_{TLB} \times f_{TLB} + W_{RegAcc} \times f_{RegAcc} + W_{MemAcc} \times f_{MemAcc} + W_{PeriphAcc} \times f_{PeriphAcc}$$

where.

$$f_{Instr} = .2 \times f_{Instr,jmp} + 2 \times f_{Instr,except} + 0 \times f_{Instr,ctrl} + 12 \times f_{Instr,coproc_{15}} + 0 \times f_{Instr,LdSt} + f_{Instr,arith} + f_{Instr,other}$$

and.

$$f_{Instr,i} = .\sum (instructions\ of\ type_i\ in\ k - cycles)$$

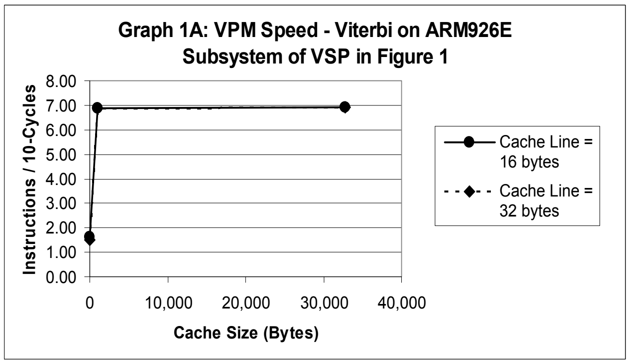
variability and variability optimization functions – such as minimization of variability – are addressed as part of the experimental procedure. In the latter characteristic, simulated systems and real systems are very similar.

iv. It then remains to determine which factors effect the power, speed and cost computations and what combination of factors produces optimal outcomes. In an industrial engineering set of experiments, we would want to determine whether the optimum we had achieve was local or whether a better result could be achieved and what factors can be adjusted to produce the better outcome.

Table 3: Factors Determining the Number of Experiments to be Peformed

Factors	Variants	Number of Variants	Number of Experiments
I-cache	Enabled, disabled	2	2
I-cache size	1k, 4k, 8k, 16k, 32k, 64k, 128k	7	7 * above = 14
I-cache Line Size	16B, 32B, 64B, 128B	4	4 * above = 56
D-cache	All variants – as for I cache	56	28 * above = 3,136
TLB	32, 64, 128 entries	3	3 * above = 9,408
I & D Bus Width	4B, 8B, 16B	3 * 3	9 * above = 84,672
I & D Memory	1st R/W = 4, 5, 6, 8 2nd R/W = 1, 2 (DDR, SDR)	2 * 4	8 * above = 677,376
Target code (programs)	Linux, Viterbi, Sieve	3	3 * above = 2,032,128
Event Weights in Table 2	Ibase, ijmp, …..	∞	∞
Etc.			

The design of experiments methodology relies on the ability to vary several variables in the system being observed in order to calculate the effects of the variables and the interactions between variables in terms of the objective functions. The prioritization of variables and interactions that cause the greatest effects gives us a handle by which to choose values of variables



that guarantee an optimal outcome. If there are no interaction effects between variables, the response of the objective function will be linear wrt the variables. Interaction effects produce higher-order polynomial responses.

Table 3 sets out the values of variables that can be set in experiments. It is impossible to perform but a small subset of the experiments in a reasonable amount of time given that simulation runs of 500 million cycles during a Linux boot might take a few minutes, in full data acquisition and profiling mode. Fortunately, nor is it necessary, the number of experiments can be reduced dramatically using fractional factorial designs in which the number of experiments is determined by the main effects and their interactions.

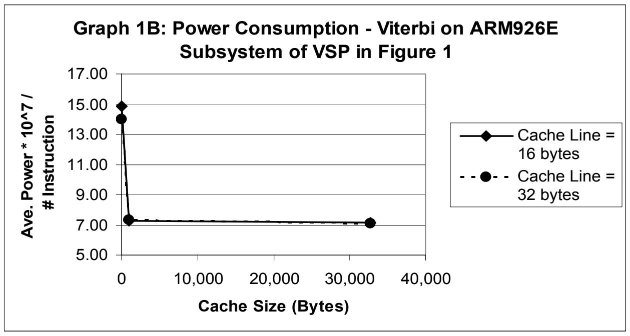
In our study, we ran exploratory experiments using Viterbi and Linux target code on many model variants and assessed the patterns of results in the light of analysis and expected behaviours. This preliminary investigation indicated that the important main effects were: I&D cache enabled/disabled; I&D cache size – 1k and 32k, cache line size – 16B & 32B, data rate of memory (DDR – double data rate, SDR – single data rate, and code), and target code. For simplicity here, we ignored interaction effects, even though to reach a global optimum they are likely to have an impact.

5. Experimental Results

We constructed 4 sets of experiments (58 in total) using various code running on the VaST ARM926E-based VSP subsystem with instruction and data buses bridged to a shared memory. The VSP subsystem was extensively parameterized and we used various configurations of cache and memory. For all experiments, the speed performance is calculated as instructions executed per 10 cycles (IPC_{10}) on the VSP (that is it is an index of VSP speed NOT processor speed) and power consumption is a relative measure of average power over all instructions executed.

5.1 Viterbi

The results from 7 Viterbi (calibration) experiments are expected, see Graphs 1A & 1B. Uncached performance is poor both in regard to power consumption and speed (IPC_{10}). With cache enabled, and even minimal cache (1,024 bytes) is sufficient, a good working set fit of Viterbi to cache was achieved. If the ARM926E was the selected controller



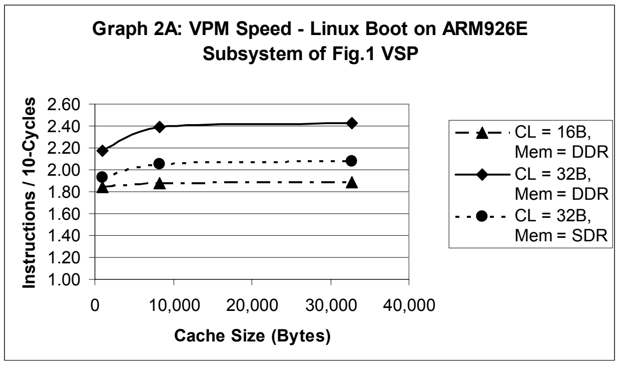
implementing an acoustic filter then a cache size of 1k bytes would be adequate. Since there is a better than 99.5% hit rate on the D-cache and I-cache, cache line size is immaterial as is bus width and memory type (either DDR or SDR). However, to minimize cost, SDR memory would be used instead of DDR.

Optimizing Objective Functions:

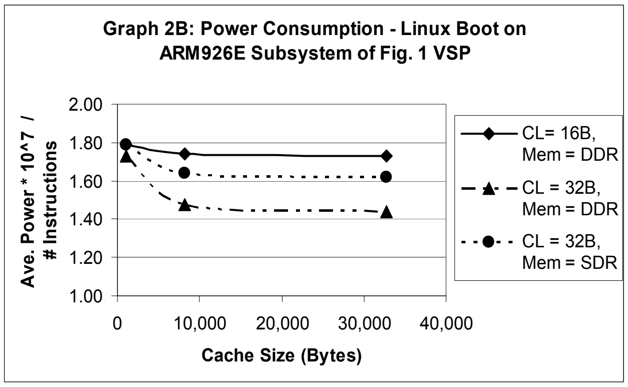
Generalizing the results – for target code with a working set size that matches the cache size, cache size is the dominant determinant in optimizing speed and power consumption in the single processor VSP subsystem. When the optimum cache is the smallest selectable, cost is also minimized with respect to this factor. Depending on the overall system objective function f_{System}*(Power, Speed, Cost)* the selection of optimal sets of settings (the so-called optimal response contour) will be determined by the tradeoffs inherent in the objective function.

5.2 Linux Boot

The Speed (IPC_{10}) and relative Power Consumption of 9 structural variants of the experimental VSP were computed while booting Linux. The variants were selected from the full set of variants determined by - cache size: 1k, 8k, 32k; cache line: 16B, 32B; Mem configured as DDR (1st word delayed 5-cycles, 2nd word available per ½ cycle) and SDR (1st word delayed 5-cycles, 2nd word available per 1 cycle); bus data width 4bytes. The results are shown in the Graphs 2A & 2B.



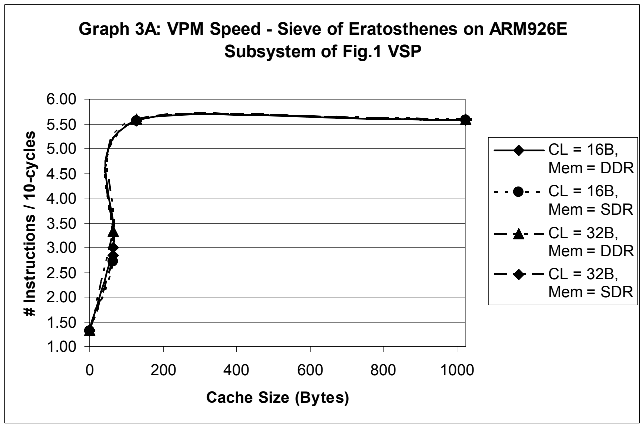
The boot sequence of Linux spends more than 50% of its time executing with the ARM926E I&D caches disabled. Linux performs initialization of the cache after the Initial Program Load, kernel load and the device driver installations. Once the operating system has booted and the idle loop is executing, the behaviour of the ARM926E VSP is much the same as its behaviour running Viterbi – that is the working-set size is compatible with any cache size. As is also expected, in an environment where the working set size of the target code greatly exceeds the cache size, the impact of the memory hierarchy on power and speed is considerable. For booting Linux, the settings of the ARM926E VSP subsystem: cache size (32 kbytes), cache line size (32bytes), and Memory (DDR) yield minimum power consumption and maximum VSP speed.



To mimimize cost, as well, a cache size (I&D) of 16 kbytes would proportionally reduce silicon cost by about 30% and adversely affect both power and speed by about 1%. To further optimize for cost, cache sizes of 8 kbytes will yield a further ~25% reduction in silicon with a worsening in power consumption and speed of 5%-10%..

5.3 Viterbi Executing on Linux

If the target code workload is Viterbi executing instead of the *Idle Loop* of Linux then the analysis in Section 6.2 remains valid. This is far from a representative workload for a general purpose computer but it may easily be a representative of the constrained workloads on embedded processors – especially those executing real-time control code.



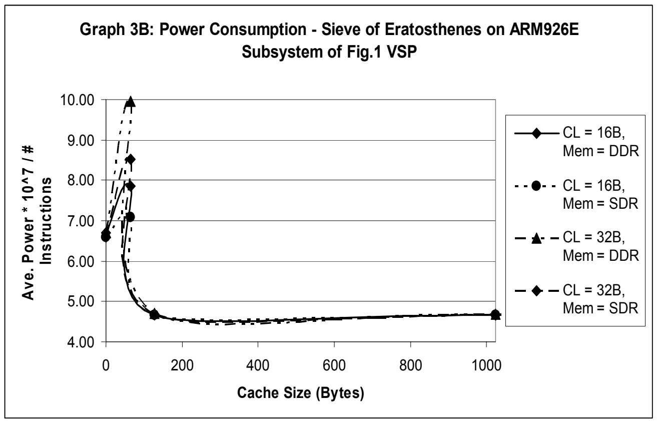
For real-time systems, a requirement is to demonstrate the meeting of service deadlines. A simple experiment to refute the hypothesis that the VSP will not meet the deadline, is to set worst case delays for appropriate peripheral devices attached to the VSP, then run the experiment. For the simple VSP used here, memory being set as DDR or SDR gives the flavour of the experiment.

5.4 Alternate Memory Hierarchies

This investigation considered a pure embedded systems problem, that of finding the best tradeoff between speed, power consumption and silicon cost for a controller executing a limited amount of code – a prime number generator using the sieve of Eratosthenes algorithm [4]. This has the same outcome as the Viterbi experiment for cache sizes above 1 kbyte. However, we were interested in this experiment in determining the near minimum cache size that would still yield within 5% of optimum speed and power for the VSP .

In this experiment we considered I&D cache characteristics: size of 0B (uncached), 64B, 128B, 256B, 1 kB, 4 kB and 8kB; cache line size (16B, 32B), wayness (1, 2, 4), cache power weighting (3, 4, 5 – depending on size) and memory type (DDR, SDR). We varied the relative power consumption of the cache based on size. The results of the experiments are shown in Graphs 3A & 3B. The speed of the VSP followed expectations except that the transition between 64 bytes and 128 bytes was sharp and at 128B the VSP essentially achieved full speed. The power graph shows another picture. Uncached power consumed by the VSP was 20% - 35% less than the power consumed by 64 byte caches (variability was due to cache line size, wayness and memory type) and 200% higher than power consumed with 128 byte caches. What we are observing here is the step-function effect on power consumption of installing a cache in a processor. For the sieve program, beyond 512 bytes, the power consumed was stable and about 20% higher than the minimum cache configuration at 128 bytes.

The effect on power consumption of installing a small cache in a processor to achieve a 4-fold increase in performance has a detrimental effect on power consumption due to the infrastructure required to support the cache. The cost of a cache is also high since the infrastructure consumes relatively large silicon real estate. These considerations led to an investigation



of alternative memory hierarchies that might achieve a better trade-off between speed, power and cost for a controller running a limited amount of code in an embedded application.

We varied the cache_hit/miss power weightings of the processor (see Table 2) to mimic the relative power consumed by a dedicated external buffer of 128 bytes (essentially a small, physically addressed, direct-mapped, on-chip cache external to the processor). This architecture is similar to the buffer organization found in processors like the Renesas SH2A [6] a processor popular in automotive control [1] where differences of cents in the price of a controller translate to several million dollars in large manufacturing runs. The results were that we could achieve a further ~40% power saving whilst maintaining near optimum speed. The cost of the chip is close to the non-cache cost. To prove that this is a global minimum requires more sophisticated statistical machinery (see [3]).

5.5 Algorithm Optimization

The final 10 experiments considered the effect of alternate algorithms on the problem of optimizing a VSP (software + hardware) for a particular (embedded) application. Since we had good empirical data already for the sieve prime number generator, we acquired from the web Kazmierczak's prime generation algorithm [7] and used the same experimental set-up as for the sieve experiments. The Kazmierczak algorithm required a small external buffer of 512 bytes to achieve maximum speed (IPC_{10}) ~40% higher than sieve and power consumption ~15% higher than sieve.

Clearly, algorithms have a 1st order effect on power, speed and cost – often say the dominant order effect! By just looking at or mathematically analyzing both the sieve and Kazmierczak algorithms, it is inconceivable that the optimal VSPs – that is software-hardware structure, as determined in this paper, would have been discovered.

6. Discussion and Conclusions

Empirical experimentation is a powerful mechanism with which to refute hypotheses that, when carefully constructed, drive the quantitative engineering process. To engage in this engineering process, prior to the existence of a physical realization, requires the existence and use of a model. If hypothesis building concerns speed, power consumption, reaction time, latency, meeting real-time schedules, etc. the model needs to be timing accurate (processor, buses, bus bridges and devices). If the extensive execution of software is an intrinsic part of the empirical experimentation, then the model needs to have high performance across all components. This paper assumes the existence of pre-silicon, high performance (20-100 MIPS), timing accurate virtual system prototypes.

Optimizing systems with complex objective functions is not intuitive. Complex tradeoffs between hardware structure and the software and algorithms that are executed on the hardware cannot be done by ratiocination or formal analysis alone, the acquisition of data as part of well-formed experiments refuting thoughtfully constructed hypotheses (ratiocination) enables decision making driven by results. Optimization comes from considering hardware and software together – not separately.

7. Acknowledgement

The capability underlying the empirical experimentation and data gathered and used to write this paper is due to the vision, effort and extra-ordinary dedication and execution of the high caliber VaST R&D team. Primus inter pares are Neville Clark, James Torossian, Foo Ngok Yong and Patricia Hughes. It is a pleasure to work with the whole VaST team.

8. References

[1] Winters, F.J., Mielenze, C. and Hellestrand, G.R. Design Process Changes Enabling Rapid Development. Proc. Convergence 2004 P-387, Oct 2004, 613-624, Society of Automotive Engineers, Warrendale, PA.

[2] Hellestrand, G.R. The Engineering of Supersystems. IEEE Computer, 38, 1(Jan 2005), 103-105.

[3] Hellestrand, G.R. Systems Architecture: The Empirical Way – Abstract Architectures to 'Optimal' Systems. ACM Conf. Proc. EmSoft2005, Sept 2005, Jersey City, NY.

[4] EEMBC: Embedded Microprocessor Benchmark Consortium. www.eembc.org

[5] Anthony, J. Design of Experiments. 2003, Butterworth-Heinemann, Oxford, UK.

[6] Renesas SH-2A, SH2A-FPU Software Manual, Rev 2.00, REJ09B0051-0200O, 13 Sept. 2004, Renesas Technology, Tokyo, Japan.

[7] Kazmierczak, M. Simple method of finding primes. 2002. http://www.mkaz.com/math/primes.html

[8] Lee, E. Absolutely Positively on Time: What Would It Take?. IEEE Computer, 38, 7(Jul 2005)

[9] Heiser, G. Private communication. Sept 2005. http://ertos.nicta.com.au/Research/L4/