CACTI 4.0

David Tarjan, Shyamkumar Thoziyoor, Norman P. Jouppi
HP Laboratories Palo Alto
HPL-2006-86
June 2, 2006*

cache modeling, area, delay, power, leakage power

The original CACTI tool was released in 1994 to give computer architects a fast tool to model SRAM caches. It has been widely adopted and used since. Two new versions were released to add area and active power modeling to CACTI. This new version adds a model for leakage power and updates the basic circuit structure and device parameters to better reflect the advances in scaling semiconductors which have been made in the last ten years. A graphical web interface has also been added to make CACTI more accessible to a larger potential audience.

* Internal Accession Date Only
Approved for External Publication

CACTI 4.0

David Tarjan, Shyamkumar Thoziyoor, and Norman P. Jouppi
dtarjan@cs.virginia.edu, sthoziyo@cse.nd.edu, norm.jouppi@hp.com

Abstract

The original CACTI tool was released in 1994 to give computer architects a fast tool to model SRAM caches. It has been widely adopted and used since. Two new versions were released to add area and active power modeling to CACTI. This new version adds a model for leakage power and updates the basic circuit structure and device parameters to better reflect the advances in scaling semiconductors which have been made in the last ten years. A graphical web interface has also been added to make CACTI more accessible to a larger potential audience.

I. INTRODUCTION

Since its release in 1994, CACTI has become an indispensable tool for many computer architects. It is used directly in most publications when evaluating memory structures implemented inside a microprocessor, as well as indirectly as a component in a temperature simulator [2] and a widely used power simulator [1].

However, several aspects of CACTI 3.2 have recently limited its applicability. First, the technology scaling model in previous versions of CACTI is simply ideal linear scaling. However many aspects of deep sub-micron technology scaling are far from ideal. Second, CACTI was originally developed in an era when 64KB caches were considered very large, whereas on-chip caches of tens of MB are now common. The original circuits modeled in CACTI were not designed to cover such a large range. Third, the power model in CACTI only considered dynamic power, but leakage power is now approaching or exceeding the level of dynamic power in many caches. Addressing these and other issues required a major revision to CACTI.

II. UPDATES AT THE PROCESS AND DEVICE LAYER

A big change in fabrication processes since the introduction of CACTI 1.0 in 1994 has been the introduction of copper interconnects and low-k dielectrics. These lower the resistivity and capacitance of wires respectively. To reflect the use of copper, the new version of CACTI lowers the resistivity of wires to 2/3 that of the old model. This reflects both the lower resistivity of copper itself as well as the diffusion barriers which isolate copper wires from the rest of the chip. Low-k dielectrics with K values as low as 2.9 have been announced in volume processes [3], so the wire capacitance was lowered by a factor of 2.9(low-k)/3.9 (SiO2). We also used data from a high-performance 130nm industrial process [4] to recalibrate the capacitance per unit length of all wires.

The last 10 years have also seen a multiplication in the number of wiring levels usable by signals. A general division between local/intermediate and global wires has emerged, with each level having wires with better resistance and capacitance, at the expense of density. While we assume that global wires are reserved for the clock tree and communications between functional units, intermediate wires still offer

advantages for long wires inside a functional unit such as a cache. Using data from the ITRS roadmap [[5], the intermediate wires are assumed to have 1.4 times less capacitance per unit length and only half the resistance per unit length than local wires. Intermediate wires are now used for all wires outside of the subarrays in CACTI 4.0, trading off a little extra area for extra speed or less power.

Even with all the advances in materials that we have enumerated, wires have not kept up the same scaling as devices. They have progressively become slower relative to transistor switching speeds [22]. In order to capture this scaling trend, in version 4.0 we make the assumption that sheet resistance scales linearly with feature size.

Another change in logic processes has been tighter wire spacing relative to device sizes. The original CACTI assumes that local wires have a pitch of 5 times the features size. This has been changed to 3.2, to better reflect modern processes.

III. UPDATES AT THE CIRCUIT LEVEL

One of the main metrics by which a SRAM is judged is the size of the individual cell. CACTI used a cell which was laid out by hand for a generic 0.8µm CMOS process for version 1.0. It has a width of 10F and a height of 20F, where F is the feature size of the process. SRAM cells in modern processes are substantially smaller than that, making CACTI overestimate the area, and thus also the delay and power of a cache in a modern process. When looking at real designs, each manufacturer has their own SRAM cell design, optimized to the market segments they are serving [6][7][8]. We used an average of SRAM cell sizes published in the last several years by manufacturers as well as the 2004 ITRS roadmap update [5] to arrive at a cell size of 120F^2. This is 60% of the original cell size and leads to a concomitant shortening of wires in the overall cache. Figure 1 shows a comparison of areas obtained from versions 3.2 and 4.0 of CACTI for various cache capacities in 70 nm technology. Other organizational parameters of the cache configurations used in creating this figure may be found in Appendix B. From Figure 1 it can be seen that because of the updates carried out at process and circuit levels, and mainly because of reduction in SRAM cell size, version 4.0 produces lower values for area compared to version 3.2.

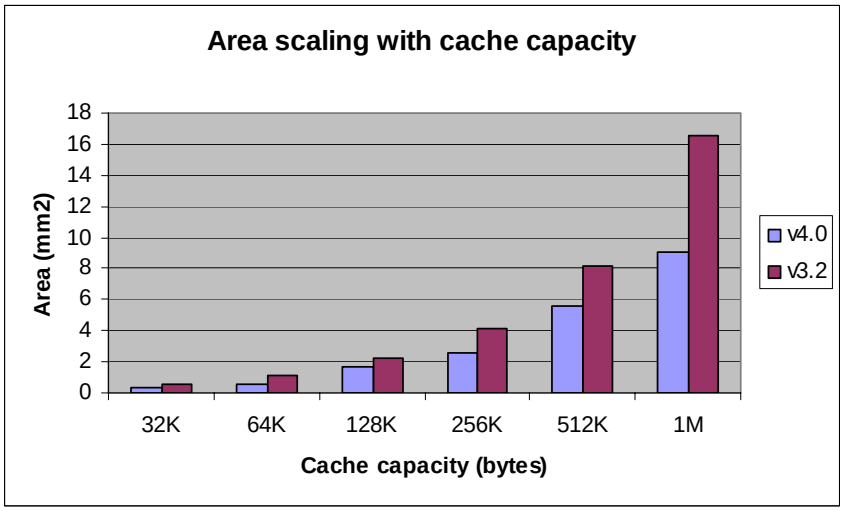

Figure 1: Scaling of area with cache capacity for 70 nm technology. Because of the circuit and process updates, version 4.0 produces area values that are lower than that produced by version 3.2.

In order to study the effectiveness of the updates to the area model, we compare the areas values obtained from CACTI 4.0 with either published area values or area values measured from die photos for three industrial cache designs from recent ISSCC papers [9][10][11]. Table 1 shows the results of this comparison. It can be seen that the average of the magnitudes of the error values is about 17%, which is quite good considering that the CACTI cache model is relatively generic. Note that CACTI does not model parity bits and so in some cases the impact of parity bits was ignored. Wherever possible the area contributions due to ECC and redundancy were calculated and subtracted from the measured/published cache areas. The descriptions of the cache designs and the parameters that were used as input to CACTI are presented in Appendix A.

Cache design	Measured/Published area (mm^2)	CACTI 4.0 area (mm^2)	Error (%)
130 nm Itanium2 6MB L3[9]	195.7	222.6	13.7
180 nm Itanium 3MB L3[10]	170.9	179	4.7
90 nm SPARC 4MB L2[11]	128.7	86.5	-32.8

Table 1: Comparison of area values obtained from CACTI 4.0 with either published area values or area values measured from die photos for three industrial cache designs from recent ISSCC papers.

The gate sizes in CACTI up through version 3.2 have been fixed except for the wordline drivers. This meant that they were not necessarily optimized for any particular design goal, but just a compromise for the expected range of sizes and organizations. Incorporating and extending modifications made in eCACTI [12] from University of California, Irvine, we calculate the widths of almost all gates based on the capacitative load they are driving, optimizing for minimum latency and good power efficiency.

In CACTI 3.2 a large percentage of the modeled cache power consumption was in the sense amplifiers. To model real caches in a better manner CACTI 3.0 had actually modeled lower power consumption for the current mirror based sense amplifier circuit by making the assumption that the three foot transistors are enabled only after the word line reaches the threshold voltage of the pass transistors that control the read operation of the memory cell. The foot transistors were then disabled when the bitlines returned to the precharged state. This reduced the power consumption of the sense amplifiers in CACTI 3.0 compared to earlier versions, however is spite of this change the sense amplifiers continued to account for an excessive percentage of power consumption. In order to reduce the power consumed in the sense amplifiers further, in version 4.0 we have updated the sense amplifier circuit to one that is commonly used in real designs and which consumes lower power than the original current mirror based sense amplifier. Figure 2 shows the new latch-based voltage-mode sense amplifier that has been assumed in version 4.0; for a discussion of the operation of this sense amplifier please refer to [13][14]. We assume that the sense amplifier enable signal is generated using a self-timed approach that reduces its static power consumption. For versions 3.2 and 4.0, Figure 3 shows the percentages of power consumed in the sense amplifiers and in the rest of the cache during a read access of a 32KB cache (other organizational parameters of the cache may be found in Appendix B). It can be seen that in version 3.2 sense amplifiers account for about 62% of the total power consumed in the cache whereas in version 4.0 they account for less than 1% of the total power.

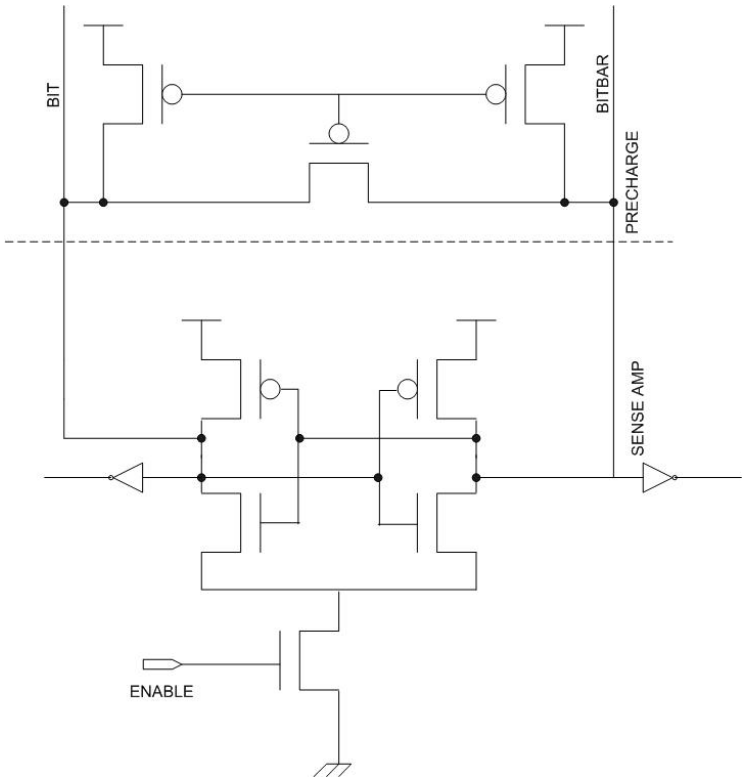

Figure 2: Latch-based voltage mode sense amplifier. This sense amplifier design consumes lesser power than the original CACTI current mirror based sense amplifier design

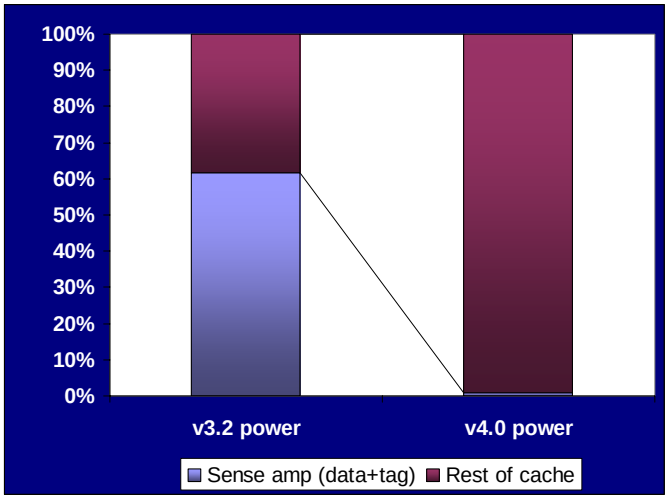

Figure 3: Power consumption in sense amplifiers of a 32KB cache for versions 3.2 and 4.0. Because of the circuit update in version 4.0 the power consumption in the sense amplifiers is much lower compared to that in version 3.2.

IV. Organizational Changes

The increase in main memory sizes has lead to the fact that today's 64 bit machines have to be able to address much more than just 4GB. While some high-end processors support a 50 bit physical address space [15], the current dominant mainstream consumer processor supports a 36 bit physical address space [17]. As a compromise, we settled on 42 bits for the default physical address space. This has the effect of increasing the size of the tags and tag comparators (we assume physically tagged caches, since virtually tagged caches and 64 bit address spaces would lead to excessively large and slow tag comparators). To optimize the tag comparators for speed, the circuit was split from two half-comparators to four quarter-comparators.

One consequence of longer tags is that the wordlines in the tag array also increase proportionally. While previous versions of CACTI never divided the global tag wordline into segments, this has been changed for the current version. This change mostly helps highly associative caches, which have a larger number of tags mapped to a single wordline.

One of the consequences of poor wire scaling has been to expose one of the design choices of CACTI 1.0 as nonscalable. The original CACTI used a single buffer driving the predecode bits to all the subarrays in parallel along horizontal lines between each pair of rows of subarrays in parallel. This model was changed in version 3.0 to an H-tree, still driven by a single driver. This model was clearly neither scalable when increasing the size of a cache nor when scaling to smaller feature sizes. In version 4.0 we replaced the H-tree that had just a single global driver with an active H-tree. At each branch of this active H-tree, a new driver is inserted which makes it scalable and saves both power and time. Similar improvements were also made for the circuits and wiring involved in driving the way select signal to the output muxes and the circuits driving the final output to the edge of the cache.

To more accurately model modern caches, some extra bits were added to each tag entry. Previously CACTI modeled 1 dirty bit and 1 valid bit per cache line. An obvious addition is extra state for LRU replacement. Because many high-performance caches don't implement full LRU, we settled for the minimal solution of adding a single bit to each tag for most recently used (MRU). We also added 2 bits to account for the fact that most modern processors support at least a 4 state cache coherency protocol.

Previous versions of CACTI have tended to overemphasize access time versus area and power by generating configurations split into more subarrays than is common in practice. In CACTI 4.0 we have introduced a minimum memory mat size, with a default setting of 8KB. This prevents any configurations with subarrays of less than 8KB from being generated. This can slightly increase the access time of small caches and memories in comparison to previous versions, but brings significant area and power savings.

We have also noted confusion among users on the interaction between ports and banks in CACTI. Previous versions treated both parameters independently: a 2-port 4-bank cache would generate four independent memory blocks (banks) each with circuitry to implement two independent ports. In practice, we found most users expect multiported multibank memories to first synthesize dependent (banked) ports from independent banks, and only multiport the banks themselves if the required number of ports exceeds the number of banks. Thus in CACTI 4.0, if the requested number of ports is equal to or less than the number of banks, we assume each bank only implements one read/write port.

Figure 4 shows a comparison of the access times obtained from versions 3.2 and 4.0 of CACTI for various cache capacities in 70 nm technology. Other organizational parameters of the cache configurations used to create this figure may be found in Appendix B. It can be seen that because of the various updates at process

and circuit levels, version 4.0 produces lower access times for the larger caches than version 3.2. It can also be seen that as the size of the cache gets larger the access times produced by version 3.2 starts to grow at a faster rate because of the quadratic growth in wire delay. In version 4.0, this growth is no longer quadratic because of the improved scaling that has been implemented. The trend produced by version 4.0 better models the scaling behavior of real cache designs.

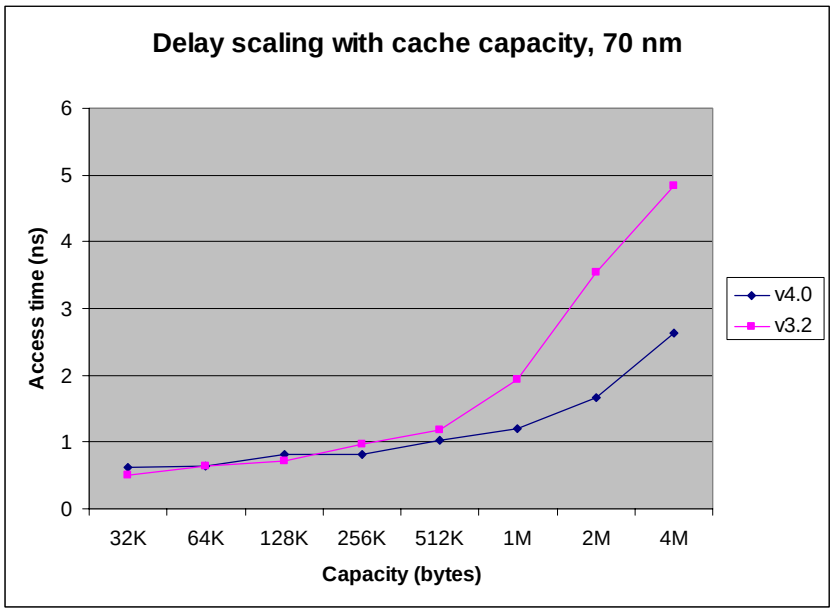

Figure 4: Scaling of access time with cache capacity for 70 nm technology. The trend produced by version 4.0 better models the scaling behavior of real cache designs.

V. LEAKAGE MODELING

Leakage current has become an important new source of power consumption for caches in deep submicron designs. For large level 2 and level 3 caches with low activity factors, leakage dominates all other sources of power consumption. To model leakage CACTI 4.0 leverages work done by the eCACTI [12] and HotLeakage [16] projects. HotLeakage models subthreshold and gate leakage based on the BSIM3 leakage equations. The leakage model for a single transistor is given by the following equation [16] (please refer to [16] for a description of the terms):

$$I_{leakage} = \mu_0 \cdot C_{OX} \cdot \frac{W}{L} \cdot e^{b(V_{dd} - V_{dd0})} \cdot v_t^2 \cdot \left(1 - e^{\frac{-V_{dd}}{v_t}}\right) \cdot e^{\frac{-|v_{th}| - V_{off}}{n \cdot v_t}}$$

In CACTI 4.0 the leakage current is calculated at 100°C, the maximum operating temperature for most commodity circuits. CACTI works by calculating the leakage for each gate after the width of the gate has been determined. The basic leakage per gate is then multiplied by the width of the circuit and then by the number of instances of that particular circuit. The leakage modeling methodology is similar to that of eCACTI and the details can be found in [12].

Figure 5 shows a comparison of the power consumption values obtained from versions 3.2 and 4.0 of CACTI for various cache capacities in 70 nm technology (organizational parameters of the caches may be found in Appendix B). Note that Figure 5 shows power consumption values for a read access. For version 4.0 the dynamic, leakage and total power consumption values are plotted. For version 3.2 the dynamic power consumption is plotted, which is the same as its total power consumption since version 3.2 did not

model leakage. The impact of modeling leakage power can be seen clearly in Figure 5. For the larger caches total power consumed in the cache is almost exclusively because of leakage. Since version 3.2 did not model leakage it produces exceedingly optimistic projections for power consumption. Note that version 4.0 produces lower projections for dynamic power consumption compared to version 3.2 mainly because of the update in the sense amplifier circuit.

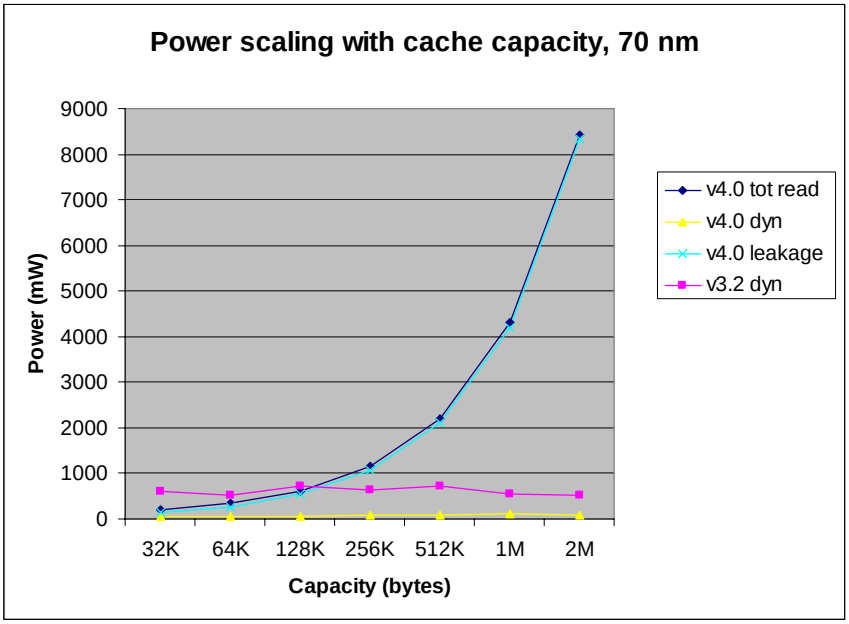

Figure 5: Scaling of power consumption with cache capacity for 70 nm technology. Version 4.0 produces power consumption values that are more realistic because of incorporation of leakage power.

VI. INCREASED PARAMETRIZABILITY

Several improvements have been made to the user interface, to make it easier to simulate a wider variety of structures more easily. For some caches it is necessary to use tags which are either shorter or longer than normal. For example, a branch target buffer has to match individual instruction addresses, not cachelines. This means that the tags for each entry are longer than for a normal cache. To accommodate this and similar cases we have added an option for the user to force the tags to be a specified length.

In some cases the user may want to only model a memory array instead of an entire cache. Examples of this include modeling a register file or specialized memory such as a queue implemented as a memory with two pointers. We now support CACTI's use for these applications with a mode that generates only a data memory without tags. Another change is that the user can now select the number of bits which are read out of the cache. The default value is 64 bits, which is a common value for L1 data caches. For instruction caches, this value could be 128 bits (4 32 bit instructions) or more, while a branch target buffer might read out 32 or 64 bits.

To save power in large high speed caches (such as on-chip L2 and L3 caches) a common approach has been to serialize the tag and data accesses. This is especially advantageous for highly associative caches, because it leads to a large reduction in the number of bitlines which are discharged. In these cases the data array of such a cache can be treated as if it were direct-mapped and can be optimized for this design point. We have added support for this kind of organization to CACTI. Figures 6a and 6b show a comparison of the dynamic read energies and access times achievable in normal and serial modes of operation for a 4-way

set associative cache in 70 nm technology. Other organizational parameters of the cache configurations used in creating these figures may be found in Appendix C. From Figures 6a and 6b it can be seen that the dynamic read energy consumed is lower when the caches are operated in serial mode at the cost of a slower access time. Note that the energy lost due to leakage in serial access mode remained more or less the same as in normal mode.

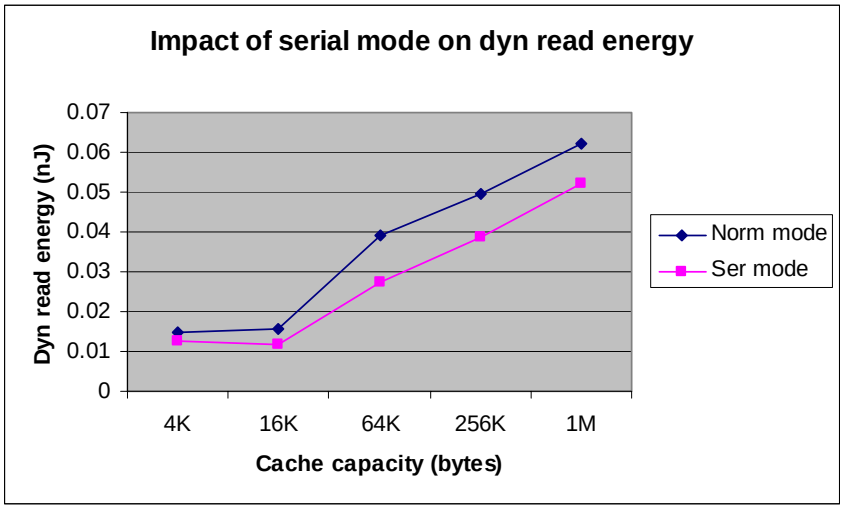

Figure 6a: Impact of serial mode of access on dynamic read energy consumption of 4-way set associative caches of various capacities in 70 nm technology. Less dynamic read energy is consumed in serial mode.

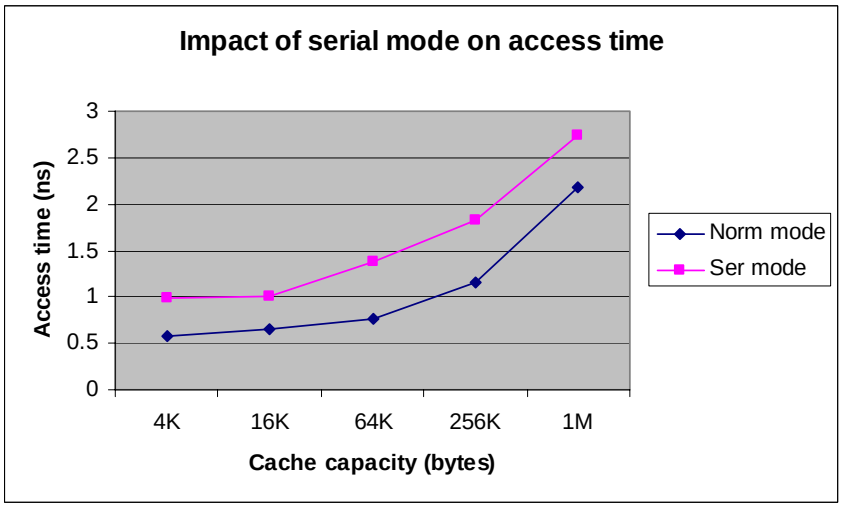

Figure 6b: Impact of serial mode of access on access time of 4-way set associative caches of various capacities in 70 nm technology. The lower dynamic read energy of serial mode comes at the cost of a higher access time.

At the other end of the spectrum, small level 1 caches still are mostly optimized for the lowest possible latency. In some cases higher power consumption for the cache is acceptable if it lowers the latency enough. For these cases we have added the option for a “fast” cache access configuration. In this case all n ways are routed to the edge of the data array and way selection is then performed at the edge between the data array and the tag array. This saves the time the way select signal needs to travel over the data array to the subarray(s) containing the n-ways. Energy per access would typically increase because n cache items

instead of a single one are put on intermediate wiring layers and sent to the edge of the data array. Figures 7a and 7b show a comparison of the access times and dynamic read energies in normal and fast modes of operation for a 32KB cache with varying degrees of associativity in 70 nm technology. Other organizational parameters of the cache configurations used in creating these figures may be found in Appendix C. From Figures 7a and 7b it can be seen that access times are lowered when the caches are operated in fast mode at the cost of an increase in the dynamic read energy consumption. Note that the energy lost due to leakage in fast access mode remained more or less the same as in normal mode.

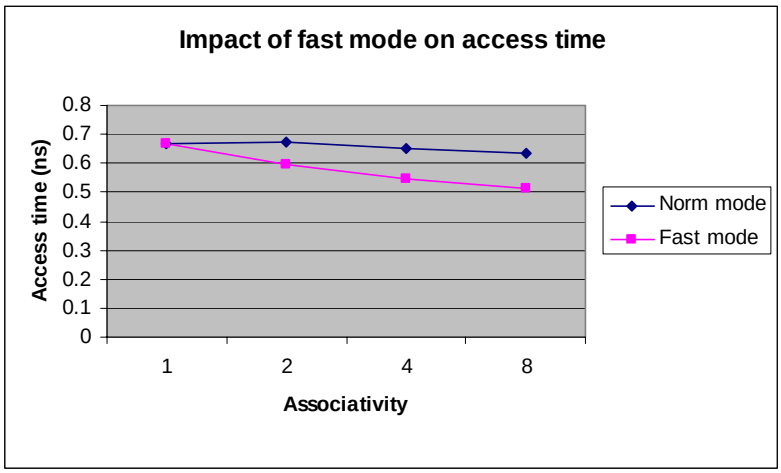

Figure 7a: Impact of fast mode of access on access time of a 32KB cache with varying degrees of associativity in 70 nm technology. Access time is lowered in the fast mode.

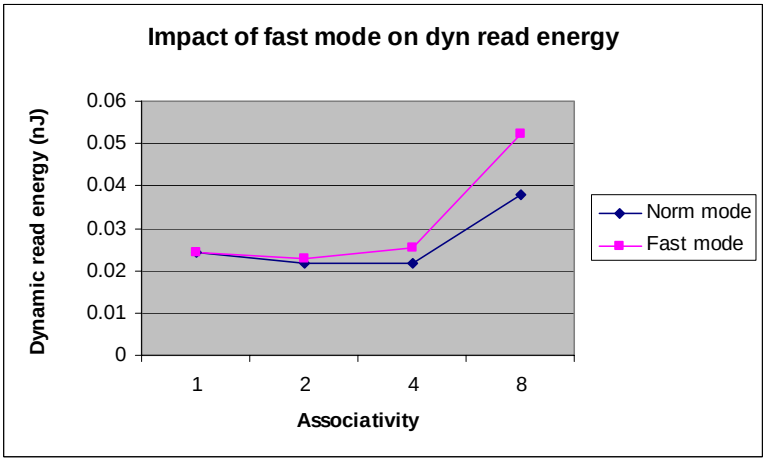

Figure 7b: Impact of fast mode of access on dynamic read energy consumption of a 32KB cache with varying degrees of associativity in 70 nm technology. The reduction in access time comes at the cost of increase in dynamic read energy consumption.

VII. INTERFACE TO SCRIPTING LANGUAGES

An outgrowth of the effort to develop a new user interface for CACTI (described in the next section), was the development of an interface layer to make CACTI useable in modern scripting languages. We used the Simplified Wrapper Interface Generator (SWIG) [18] tool/program to transform CACTI into a library

accessible from the Python runtime. SWIG is a toolset which can generate interface libraries for programs written in C or C++ to make them accessible in a wide variety of modern languages. SWIG can generate interfaces to scripting languages, such as Perl, Python, PHP, Ruby and Tcl. It also supports Java and C#. The extra code to let SWIG work with CACTI is minimal. It consists of a file *cacti.i* containing the following lines:

```
%module cacti
%{
/* Includes the header in the wrapper code */
#include "cacti_interface.h"
%}

/* Parse the header file to generate wrappers */
%include "cacti_interface.h"
```

After installing SWIG,

```
swig –python cacti.i
```

will generate a file called *cacti_wrap.c*, which can be compiled into a library *_cacti.so* / *_cacti.dll*. The naming convention is required for the python runtime to recognize the file as python module. CACTI can now simply be imported into the Python runtime and used:

```
% python
>>> from cacti import cacti_interface
>>> cache = cacti.cacti_interface(…)
```

cacti_interface.h contains both the C function cacti_interface() as well as the definitions of the data structures for configuration, timing, area and power. SWIG wraps the C structures in Python classes with getter and setter functions.

VIII. WEB INTERFACE

A side effect of the increasing functionality of CACTI in each new version has been an increase in the size of the output. In fact, the addition of leakage power has pushed the amount of output text well above the single screen mark. Finding the data point(s) which are actually important to the individual user can become somewhat tedious. One of our goals was to make the information coming out of CACTI more easily understandable.

In addition, we wanted to make CACTI usable via a web interface. Our motivation to do this was threefold:

1. A web interface would allow the central hosting of CACTI on a web server, available to anybody with access to the internet. Any changes, updates or bug fixes could be rolled out immediately to all users of CACTI.
2. Using a simple HTML form has become a normal activity in the eleven years since the first release of CACTI. Interacting with a command line UNIX utility is not something most people are familiar with. We hope that such a familiar and easily accessed interface will increase the utility and usage of CACTI and make it accessible to a larger audience.
3. A web interface would make it much easier to add graphics to the output of CACTI, something which the traditional command line UNIX environment is not readily suited for.

Our goal for any web interface was that it should be lightweight (no preexisting webserver needed) and allow easy integration with the existing CACTI code base. Thanks to the ease with which CACTI could be exposed to scripting languages with the help of SWIG, a very large number of web frameworks written in various scripting languages became viable options. We settled on a very small web application server written in Python called Snakelets[20]. On top of this server is built a blogging engine called Frog. We use Frog only for handling input/output to the web form and simple error-checking. The Python interface generated by SWIG is then called directly from Frog. The result is then passed back from CACTI to Frog for formatting and display.

As mentioned above, we wanted to make the output information from CACTI more readily understandable. Inspired by an earlier web interface for CACTI 1.0 [19] we decided that displaying the results for delay, active and leakage power as stacked bar charts would make understanding the output of CACTI easier. We first thought that we would have to generate graphics on the server for each call to CACTI, or use some form of HTML tables. Luckily we found the MultiGraph javascript library [21], which had all the features we needed. We can now generate the bar charts in the client browser with a couple of simple javascript function calls. Figures 8a and 8b show screenshots of the new CACTI 4.0 web interface and graphical output display.

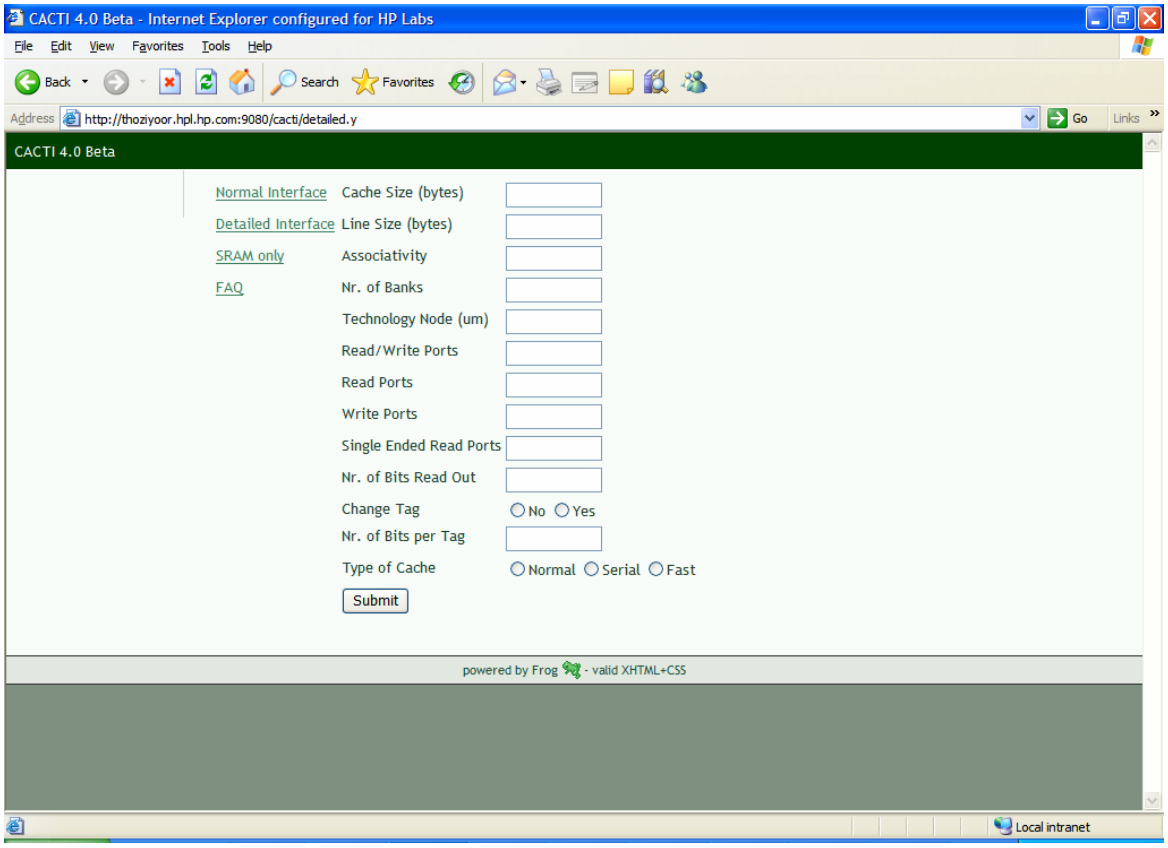

Figure 8a: Screenshot of the new CACTI 4.0 web interface.

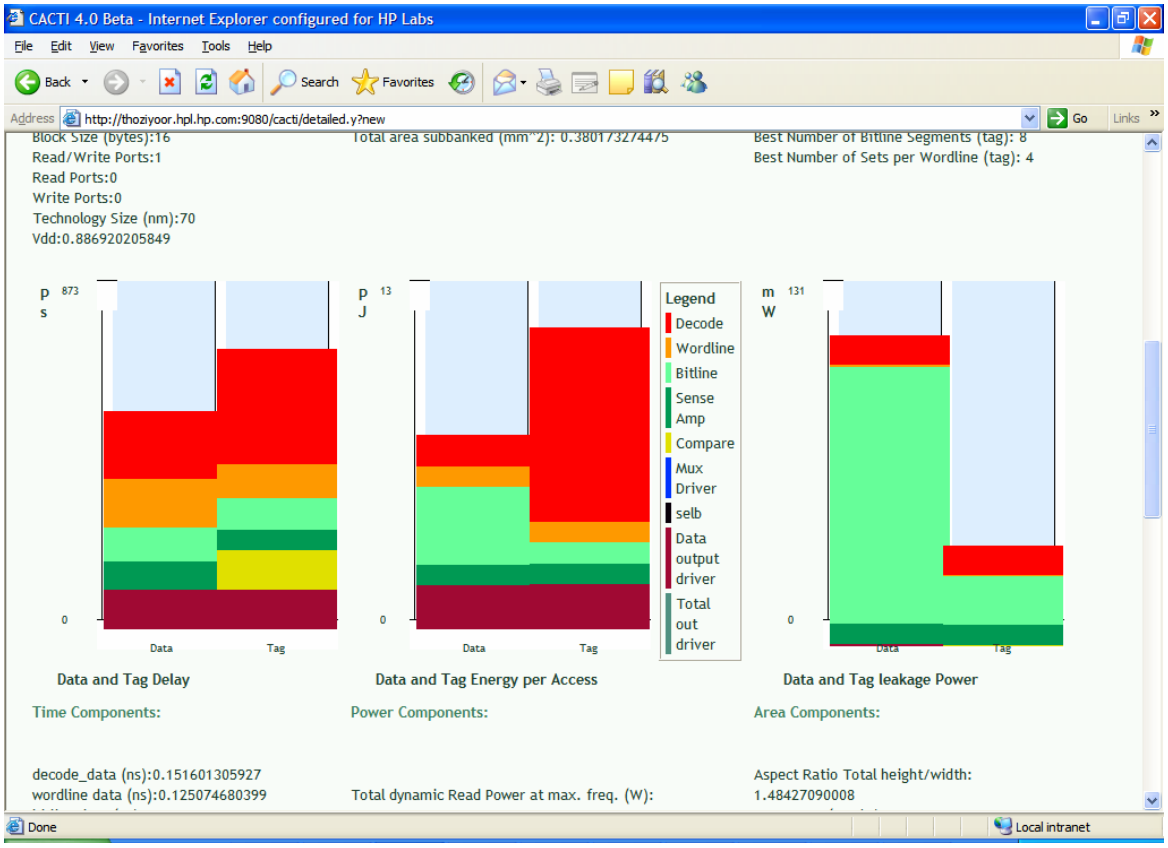

Figure 8b: Partial screenshot of the CACTI 4.0 graphical output display.

IX. FUTURE WORK

Presently CACTI models SRAM based caches. In the future we plan to extend CACTI to also model DRAM based caches in order to aid the study of various embedded DRAM based architectures. We also plan to add support for parity/ECC bits since they are an integral part of most memory structures of today.

APPENDIX A

Three industrial cache designs were used to check the effectiveness of the updates made to the area model in CACTI 4.0. These were the following:

Cache A: 130 nm Itanium 2, 6MB, L3, from ISSCC 2003[9]

Cache B: 180 nm Itanium, 3 MB, L3, from ISSCC 2002[10]; Note that for this cache we compared data array areas only.

Cache C: 90 nm, SPARC, 4MB, L2, from ISSCC 2004[11]

For each of these caches we present next the parameters that were plugged into CACTI 4.0 in order to obtain their area projections.

Cache A:

Capacity (bytes)	6291456
Line size (bytes)	128
Number of banks	1
Associativity	16 (actual is 24)
Tech node (micron)	0.13
Number of read ports	0
Number of write ports	0
Number of read/write ports	1
Number of output bits	256
Number of single-ended read ports	0
Change tag	No
Access mode	Serial

From the die photo of this cache, we measured the areas occupied by the data and tag arrays. We made an assumption that out of the 140 subarrays in the data array, 12 subarrays are being used for ECC and redundancy and subtracted the area occupied by them. Because the cache has an associativity of 24 and since CACTI only allows associativity values that are powers of 2, we specified the nearest allowable associativity value of 16.

Cache B:

Capacity (bytes)	3145728
Line size (bytes)	128
Number of banks	1
Associativity	8, 16 (actual is 12)
Tech node (micron)	0.18
Number of read ports	0
Number of write ports	0
Number of read/write ports	1
Number of output bits	256
Number of single-ended read ports	0
Change tag	No
Access mode	Serial

The published area of the data array for this cache is 175 mm^2. This value also incorporates the area consumed by 5 subarrays for ECC and 2 for redundancy. We subtracted the area spent in ECC and redundancy and then compared the resulting value with the data array area projected by CACTI. Since this cache has an associativity of 12, we ran CACTI with associativity values of 8 and 16 and then took an average of the resulting data array areas.

Cache C:

Capacity (bytes)	4194304
Line size (bytes)	32
Number of banks	1
Associativity	4
Tech node (micron)	0.09
Number of read ports	0
Number of write ports	0
Number of read/write ports	1
Number of output bits	256
Number of single-ended read ports	0
Change tag	No
Access mode	Serial

The published area for this cache is 128.7 mm^2. The data array includes 9b/16B of ECC correction as well which CACTI did not model.

APPENDIX B

Cache configuration parameters used in creating Figures 1, 3, 4 and 5:

Cache line size (bytes)	16
Number of banks	1
Associativity	1
Tech node (micron)	0.07
Number of read ports	0
Number of write ports	0
Number of read/write ports	1
Number of output bits	64
Number of single-ended read ports	0
Change tag	No
Access mode	Normal

APPENDIX C

Cache configuration parameters used in creating Figures 6 and 7:

Cache line size (bytes)	16
Number of banks	1
Associativity	4
Tech node (micron)	0.07
Number of read ports	0
Number of write ports	0
Number of read/write ports	1
Number of output bits	64
Number of single-ended read ports	0
Change tag	No

REFERENCES

[1] David Brooks, Vivek Tiwari and Margaret Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. *ISCA*, pp. 83-94, 2000.

[2] Kevin Skadron, Mircea R. Stan, Wei Huang, Sivakumar Veusamy, Karthik Sankaranarayanan, and David Tarjan. *Temperature-Aware Microarchitecture. ISCA*, pp. 2-13, 2003.

[3] S.E Thompson, et al. A 90-nm logic technology featuring strained-silicon. *IEEE Transactions on Electron Devices*, Vol. 51, No. 11: 1790-1797. Nov. 2004.

[4] S. Thompson et al. 130nm Logic Technology Featuring 60nm Transistors, Low-K Dielectrics, and Cu Interconnects. Intel Technical Journal, Volume 6, Issue 2, May 2002.

[5] Semiconductor Industry Association. International Technology Roadmap for Semiconductors, 2004 Update.

[6] Kanda M et al. Highly stable 65 nm node (CMOS5) 0.56 μ^2SRAM cell design for very low operation voltage. *Symposium on VLSI Technology*, 2003.

[7] Soon-Moon Jung et al. A novel 0.79 μ^2 by KrF lithography and high performance 90 nm CMOS technology for ultra high speed SRAM. International Electronic Devices Meeting, 2002.

[8] Nii, K et al. A 90-nm low-power 32-kB embedded SRAM with gate leakage suppression circuit for mobile applications. *IEEE Journal of Solid-state Circuits*, Vol. 39, No. 4: 684-693, Apr 2004.

[9] Jason Stinson and Stefan Rusu. A 1.5GHz third generation Itanium processor. *ISSCC*, 2003.

[10] Don Weiss, John J. Wuu, and Victor Chin. The on-chip 3MB subarray based 3rd level cache on an Itanium microprocessor. *ISSCC*, 2002.

[11] D. Wendell et al. A 4MB on-chip L2 cache for a 90nm 1.6GHz 64b SPARC microprocessor. *ISSCC*, 2004.

[12] Mahesh Mamidipaka and Nikil Dutt. eCACTI: An Enhanced Power Estimation Model for On-chip Caches. *University of California Irvine Center for Embedded Computer Systems Technical Report TR-04-28*, Sept. 2004.

[13] Jan Rabaey. Digital Integrated Circuits, 2nd edition, Prentice Hall, 2003.

[14] Sinha M, Hsu S, Alvandpour A, Burleson W, Krishnamurthy R, and Borkar S. High-performance and low-voltage sense-amplifier techniques for sub-90nm SRAM. *IEEE International SOC Conference*, 2003.

[15] C McNairy and D Soltis. Itanium 2 processor microarchitecture. *IEEE Micro*, Vol. 23, No. 2 :44-55, March 2003.

[16] Y. Zhang, D. Parikh, K Sankaranarayanan, K. Skadron, and M. Stan. HotLeakage: A temperature-aware model of subthreshold and gate leakage for architects. TR-CS-2003-05, University of Virginia, Dept of Computer Science, March 2003.

[17] IA-32 Intel Architecture Software Developer's Manual, Vol 1: Basic Architecture.

[18] Simplified Wrapper and Interface Generator, http://www.swig.org/

[19] Interactive CACTI, http://www.ece.ubc.ca/~stevew/cacti/

[20] Snakelets – Python web application server,http://snakelets.sourceforge.net/

[21] MultiGraph JavaScript library, http://www.howtocreate.co.uk/jslibs/script-multigraph

[22] R Ho, K Mai, and M Horowitz. The future of wires. *Proceedings of the IEEE*, 89(4):490–504, 2001.