FEBRUARY 2000

WRL Research Report 2000/7

CACTI 2.0: An Integrated Cache Timing and Power Model

Glen Reinman and Norman P. Jouppi

COMPAQ **Western Research Laboratory** 250 University Avenue Palo Alto, California 94301 USA

The Western Research Laboratory (WRL), located in Palo Alto, California, is part of Compaq's Corporate Research group. WRL was founded by Digital Equipment Corporation in 1982. We focus on information technology that is relevant to the technical strategy of the Corporation, and that has the potential to open new business opportunities. Research at WRL includes Internet protocol design and implementation, tools to optimize compiled binary code files, hardware and software mechanisms to support scalable shared memory, graphics VLSI ICs, handheld computing, and more. As part of WRL tradition, we test our ideas by extensive software or hardware prototyping.

We publish the results of our work in a variety of journals, conferences, research reports, and technical notes. This document is a research report. Research reports are normally accounts of completed research and may include material from earlier technical notes, conference papers, or magazine articles. We use technical notes for rapid distribution of technical material; usually this represents research in progress.

You can retrieve research reports and technical notes via the World Wide Web at:

http://www.research.compaq.com/wrl/

You can request printed copies of research reports and technical notes, when available, by mailing your order to us at:

Technical Report Distribution
Compaq Western Research Laboratory
250 University Avenue
Palo Alto, California 94301 U.S.A.

You can also request reports and notes by sending e-mail to:

WRL-techreports@research.compaq.com

CACTI 2.0: An Integrated Cache Timing and Power Model

Glen Reinman and Norman P. Jouppi

Compaq Computer Corporation Western Research Laboratory
reinman@cs.ucla.edu, norm.jouppi@compaq.com

February, 2000

Abstract

CACTI 2.0 is an integrated cache access time, cycle time, and power model. By integrating all these models together users can have confidence that tradeoffs between time and power are all based on the same assumptions and hence are mutually consistent. CACTI is intended for use by computer architects so they can better understand the performance tradeoffs inherent in different cache sizes and organizations.

This report details enhancements to CACTI 1.0 that are included in CACTI 2.0. CACTI 2.0 includes modeling support for fully-associative caches, a cache power model, technology scaling, multiported caches, improved tag comparison circuits, and other improvements to CACTI 1.0.

An Integrated Cache Timing and Power Model

Glenn Reinman Norm Jouppi
Summer Internship 1999
COMPAQ Western Research Lab, Palo Alto

1 CACTI

CACTI [6] calculates access and cycle times of hardware caches. It uses an analytical model to estimate delay down both tag and data paths to determine the best configuration for a given cache size, block size, and associativity (at $0.80\mu m$ technology size). Figure 1 demonstrates the architecture of the cache in the analytical model. In addition to providing timing data for each portion of the data and tag paths, CACTI also returns the number of data and tag arrays (in terms of the number of word line and bit line divisions), and the number of sets mapped to a single wordline, for both tag and data arrays. CACTI does not model cache area, but does estimate wire resistance and capacitance based on cache configuration.

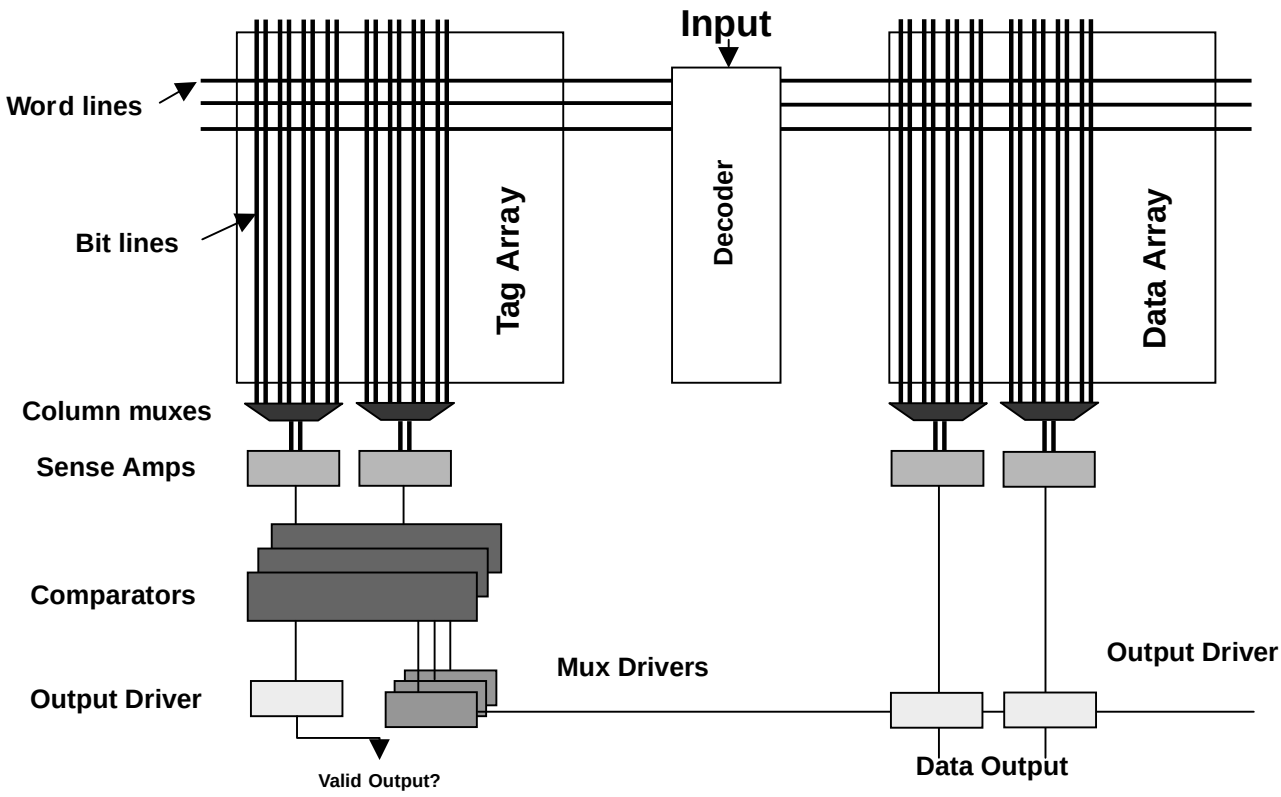

Figure 1: Cache model used in CACTI [6].

This work introduces several modifications to the CACTI model. First, the transistor widths used in the CACTI model are tuned to improve the access time and scalability of the model. Next, we address the potential

bottleneck of the tag path through a number of techniques. We also introduce several new features into the CACTI model: fully associative cache modeling, multiple cache port modeling, and cache power modeling. The timing optimization techniques, fully associative modeling, and multiple cache port modeling are described in section 2. Our power model is described in section 3. We then provide some results from the new CACTI model in section4, comparing the timing and power data to results obtained using a spice model. The syntax of the enhanced CACTI model will be demonstrated in appendix A. Sample output for our model is shown in appendix B. Finally, an overview of the files used in this project are provided in appendix C.

2 Modification to Timing Model

A number of enhancements were made to the CACTI timing model. The access times for set associative caches were optimized by scaling transistor widths and improving the performance of the tag comparison hardware. In addition, we added support for fully associative caches and for caches with multiple access ports. Finally, we changed the handling of process technology sizes and cache cycle time generation.

2.1 Transistor Tuning

Throughout the extension of the CACTI model, it was necessary to scale the width of some transistors on the critical path. Care was taken to avoid making these widths too large and wasting chip area or increasing capacitance. For the most part, the changes to transistor widths were on the tag path, especially in the multiplexor drivers. Avant! AvanWaves (version 1999.2) was used along with a spice model of the cache to determine which sections of the circuit required transistor tuning. Changes were made to both spice and CACTI models to determine the overall effect.

2.2 Improving the Tag Path

In set associative caches, the cache tags need to be checked to determine which set of output drivers to select. According to the cache model in figure 1, it can be seen that the access time of the cache is equal to

$$Max(delay_{datapath}, delay_{tagpath}) + delay_{outputdriver}$$

where the data path delay does not include the output driver. In many instances, the tag path takes longer than the data path. For example, in the original cacti model, the tag path of a 16K 2-way associative cache takes 7.8 ns, while the data path takes 4.9 ns (both excluding the data output driver).

We explored three different techniques to lessen the delay of the tag path. First, we provided the option to move the output drivers on the data path closer to the multiplexor drivers on the tag path. This decreases the delay of the tag path by reducing the load on the multiplexor drivers, but does increase the delay on the data path. This technique effectively attempts to balance the data and tag paths. Second, we looked at splitting the comparator on the tag path into two structures to reduce its latency. Third, we increased the amount of column multiplexing done on the data path prior to the sense amp stage, while decreasing the amount of multiplexing done on the data path after the compare stage. This has the added benefit of reducing the number of sense amps needed on the data path.

2.2.1 Balancing the Tag and Data Path

The output drivers on the data path were moved closer to the multiplexor drivers on the tag path, trading cache area, power, and data path delay, for decreased tag path delay. In order to drive the increased distance to the output drivers, two inverters were inserted on each sense amp output on the data path. This can be seen in figure 2. The CACTI model attempts a range of values for the scaling factor seen in the figure, and will choose the relative

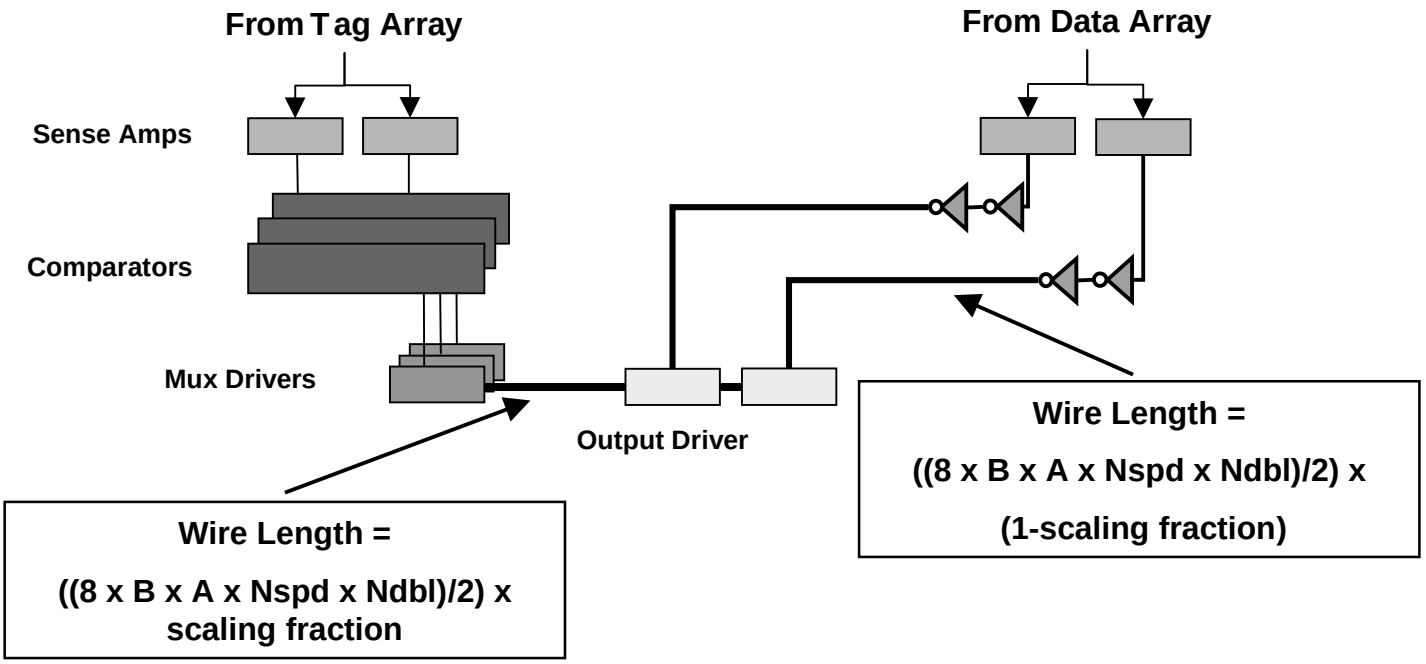

Figure 2: Balancing the tag and data paths. CACTI will determine the scaling factor for the wire length between the mux drivers and output drivers that will result in the lowest overall cache delay.

position of the output drivers that results in the smallest overall delay. If the benefit of this optimization does not outweigh the cost, the tag path will be left as before, without the additional inverters.

2.2.2 Split Comparator

The second technique involves splitting the comparator on the tag path into two smaller comparators. Each comparator handles one half of the address bits to be compared. This reduces the capacitative load on the comparison line. The two comparators can then be recombined using a NAND gate in the subsequent multiplexor driver stage. The NAND gate will replace the existing inverter used to drive the multiplexor driver. This can be seen in figure 3. The comparator is only split once, as merging more than two signals would likely prove more costly than the savings obtained by further reducing the capacitative load on the comparison line.

2.2.3 Multiplexing Shift

This final change again involves shifting more of the delay from the tag side to the data side. Originally, the multiplexors following the compare stage on the tag path would select from both the different associative entries in the data array and the possible output bits in a single cache line. For example, in a 2-way associative cache with 32 byte lines and 64 output bits, there would need to be 8 way multiplexing at this stage. The output bit selection does not depend on the tag path, and therefore can be handled by the bitline column multiplexors that lead to the sense amps. Since the column multiplexors are already responsible for converging bitlines from various subarrays, we limit the degree of multiplexing to 16 (i.e. 16 bitlines to a single sense amp) for the column multiplexor. We introduce this limitation to avoid allowing too many bitlines to share a single sense amp.

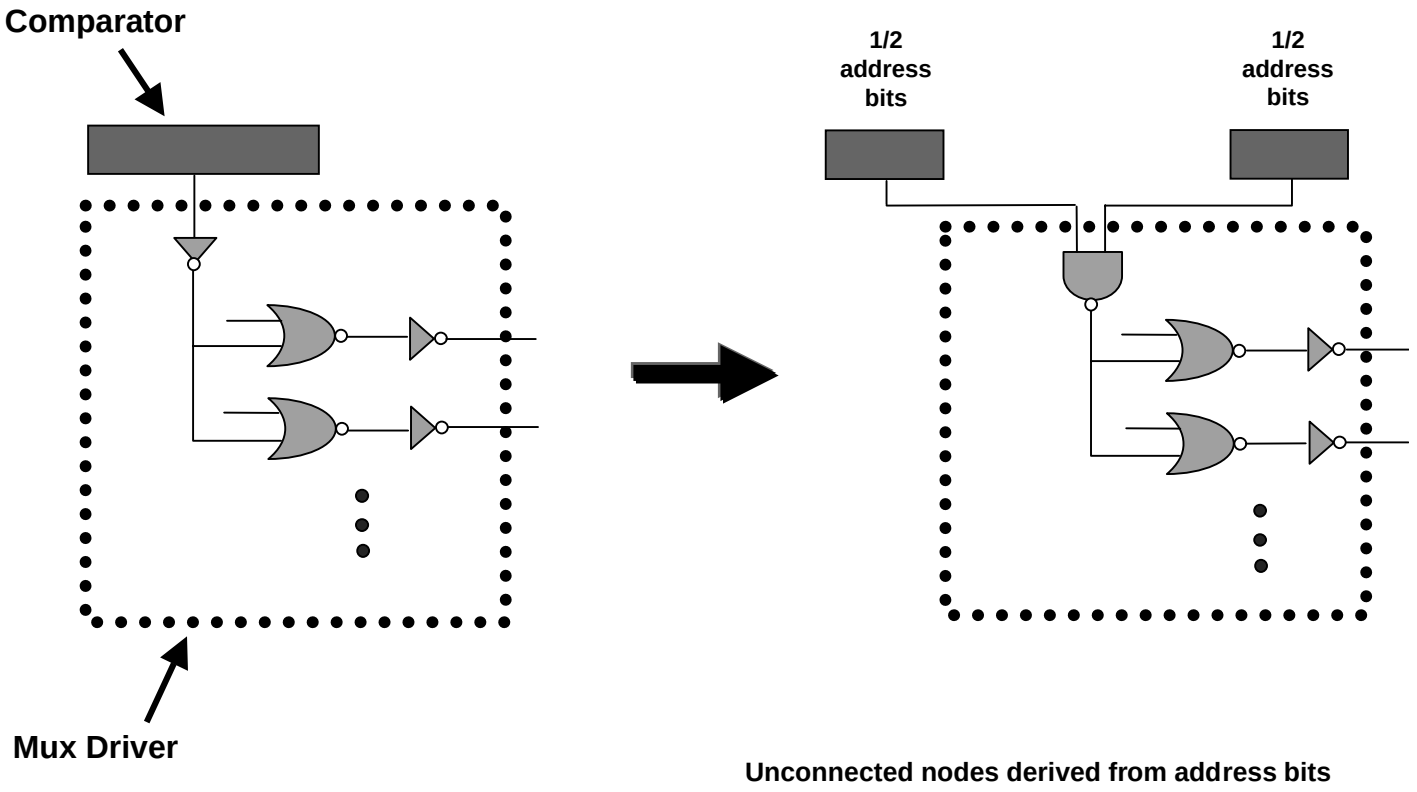

Figure 3: Illustration of split comparator. Each split comparator handles half of the address bits - each performing half of the comparisons of the original single comparator. A NAND gate replaces the inverter in the mux driver and is used to join the signals from the two halves.

2.3 Fully Associative Cache

The new version of CACTI includes support for fully associative caches. In the fully associative cache model, the customary tag path is replaced by a fully associative cache decoder. Rather than tracking a separate tag and data path, the fully associative cache has a single path. The decoder will drive the wordlines of the data array as in the original cache model, but there will only be a single cache entry associated with each wordline. In the decoder, all tag entries are checked for a match. Should a match exist, a single data array wordline will be enabled. Once the data array wordline is enabled, the data path of the fully associative cache proceeds in the same manner as the direct mapped cache. All decode and selection occur prior to the data array access (i.e. there is no multiplexor as in the set associative model). Moreover, as each wordline is associated with a single cache entry, there is no need to try different values of `Nspd`, `Ntspd`, `Ndwl`, or `Ntwl`. `Ndbl` may be varied, as dividing the bit lines will help reduce the delay associated with searching the entire tag array. Since selection occurs before the wordline is even driven, there are less bitlines brought low in a fully associative design - which helps in reducing the power consumed by the cache. However, since the tag comparison cannot proceed in parallel with the data array access, the delay of a fully associative cache is typically larger.

After a number of inverters to simulate the probe address drivers and timing chain of the cache, the first stage of the fully associative cache involves checking each bit of the probe address with each corresponding bit of the cache addresses. The tag comparison stage of the fully associative cache is split in a manner similar to the comparator in the set associative cache – each comparator only looks at half of the address bits. The two comparator halves

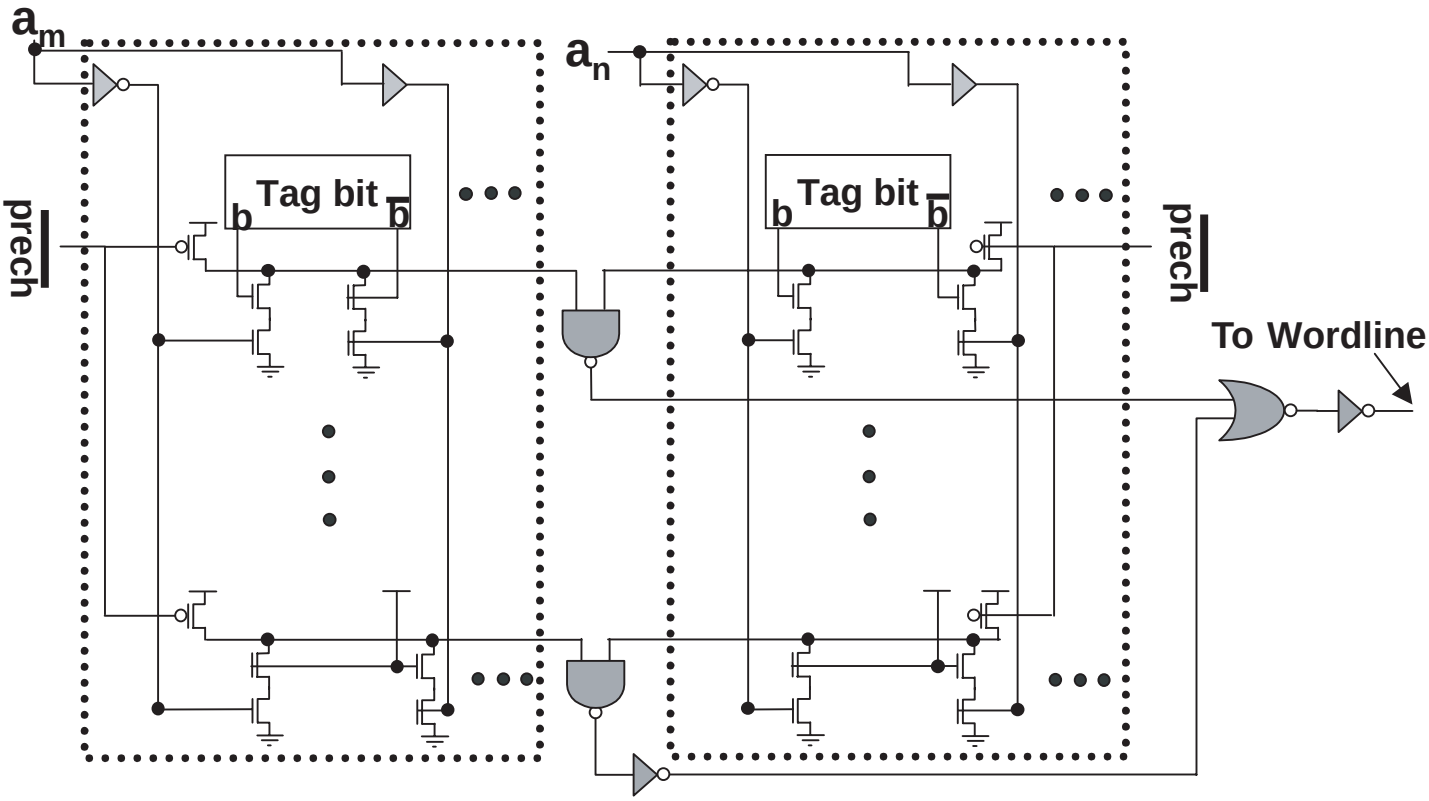

Figure 4: Fully associative cache model. Each dotted square represents a portion of one tag cell. Each tag cell handles half of the address comparisons for a particular tag entry.

are then combined via a NAND gate into a single signal. This can be seen in figure 4. Tag bits a_m and a_n belong to the probe address, and are compared, along with their inverses, to the tag bits of the cache. Each half of the comparator stage has e lines, where e is the number of entries (and the number of wordlines) in the cache subarray. Each of these lines has $x/2$ comparator pairs, where x is the number of address bits in a tag. Every address line in the comparator is initially precharged high, and if any bits in the probe address do not match the line address, the line is brought low. On a cache access, at most one line will remain high.

To maintain correct timing of the cache, a dummy line (shown at the bottom of figure 4) is used with each subarray. The dummy line has the same comparators as a regular line, but one of the comparators is fixed to bring the dummy line low when the probe address bits arrive at the comparator. Only a single comparator is fixed to pull down the address line to model the maximum delay of the address line discharge. The dummy line then passes through an inverter and is used to enable the selection of a wordline using NOR gates. Each real address line is fed into its own NOR gate which controls access to the wordline driver that corresponds to the address line. Each address line is NORed together with the dummy gate to determine when to drive the wordline. This prevents wordlines from being driven before all probe address bits have arrived. Once a wordline is driven, the data path will behave exactly as a direct mapped cache.

To model the extra space required by the tag comparison stage, the tag cell height is doubled. Tag cells for the fully associative cache are $8\mu m$ by $32\mu m$, while data cells are $8\mu m$ by $16\mu m$.

Because each address bit must travel to a comparator on every address line in the fully associative cache decoder (since we must compare the address to every tag in the cache), a tiled layout approach must be used to

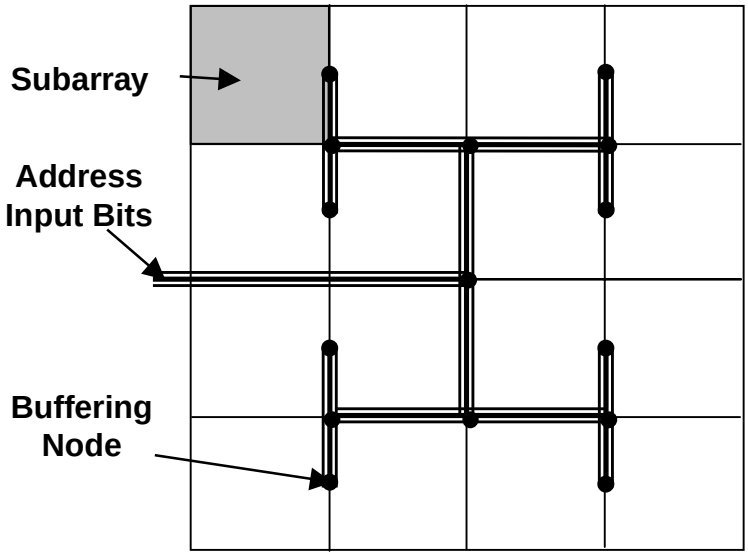

Figure 5: Layout of a fully associative cache with 16 subarrays. The address bits are brought in using an h-tree structure.

reduce the length of the incoming address bit wires. Figure 5 shows our strategy. In this figure, we tile 16 cache subarrays and route wire using an h-tree strategy. The address lines are shown meeting at several black nodes - each node represents a buffering mechanism. The tile shown in grey is a single cache subarray. Using this approach, the worst case distance to reach a subarray is reduced from n to $log(n)$.

2.4 Multiple Cache Ports

CACTI previously assumed a single read/write port on the cache model. We have expanded this model to allow the user to specify how many read/write ports (maximum of 2), read-only ports, and write-only ports to model on the cache. The extra ports are modeled as an increase to cell size, along with extra wordline and bitline lengths. All auxiliary structures (i.e. comparators, multiplexors) are assumed to be duplicated but are not included in the timing calculation. The auxiliary structures are included in the power model discussed in section 3.

Figure 6 demonstrates a three port configuration on a single RAM cell. It consists of a read/write port (Port 0), a read port (Port 1) and a write port (Port 2). If the design were single-ended, it would not require both bit lines to be added for each port, however we do not model this.

The impact of extra ports on cell size is as follows:

For each extra read port:

- increase cell size by (2 $\times$ wire pitch) in the y direction (affects bitline metal)
- increase cell size by (wire pitch) in the x direction (affects wordline metal)

For each extra read/write or write port:

- increase by (2 $\times$ wire pitch) in both the x and y directions

For example, consider the following parameters:

$$Cmetal = 275fF$$
$$Rmetal = 48mO$$

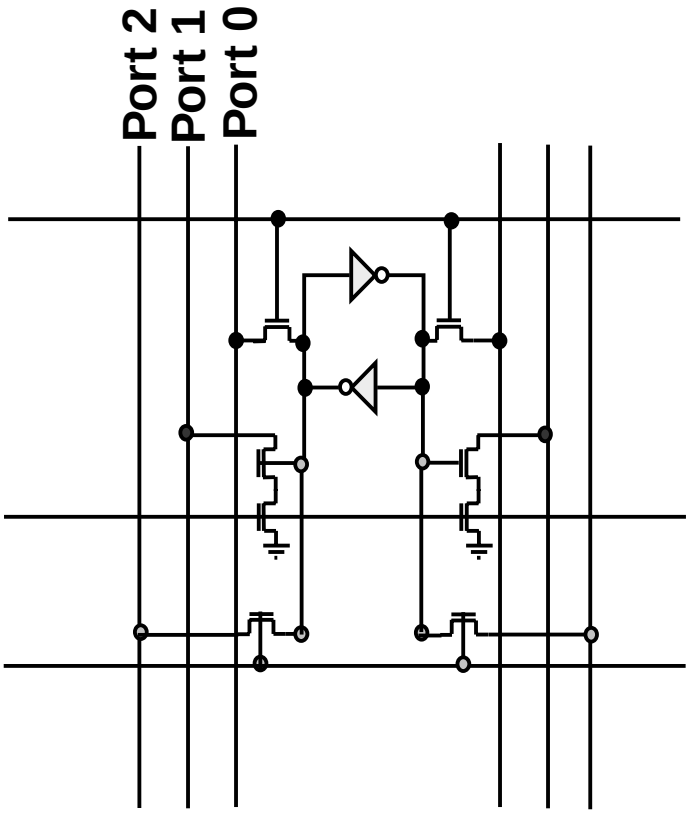

Figure 6: Multiple port example around a single SRAM cell. Port 0 is a read/write port. Port 1 is a read port. Port 2 is a write port. Each additional port impacts the cell size and wire lengths.

$$wirepitch = 4\mu m$$

A cache with a single read/write port would have the following characteristics:

$$Cellsize = 8 \times 16$$
$$C_{bitmetal} = 4.4pF$$

A cache with two read/write ports would have the following characteristics:

$$Cellsize = 16 \times 24$$
$$C_{bitmetal} = 6.6pF$$

2.5 Process Technology Scaling

As a minor tweak, the technology scaling factor already present in CACTI was made into a required command parameter. The user specifies the feature size (in microns) of the technology that is to be modeled, and the CACTI model scales measurements made at the $0.80\mu m$ size down (or up) to the desired technology size. This scaling factor affects both timing and power measurements.

2.6 Cache Pipelining

In recent years, some microprocessors (e.g., the DEC 21264[3]) have used caches that are effectively pipelined on half cycle boundaries. This can provide most of the functionality of true dual porting without the access time,

power, and area penalty incurred by true dual porting. Because the cache arrays still take a full cycle to access, there are effectively two cache accesses in the cache arrays at any given time. No intermediate latches are placed in the cache, rather the cache is pipelined using an approach similar to wave pipelining[1]. Wave pipelining uses circuits that have similar minimum and maximum delays independent of input values to keep waves of logic values separate and distinct while traveling through the circuitry. Caches are particularly well suited to wave pipelining, since their access time is largely independent of the cache line being accessed.

Wave pipelining is only possible if one logic stage does not account for approximately 33% or more of the delay through the whole circuit. In the current timing model we compute the ratio of the maximum stage delay time to the total access time. If this is less than 0.333X, we assume the cache can be wave pipelined by a factor of two. This makes the cycle time equal to half the cache access time. If this is not possible, we report the minimum cycle time possible based on the maximum stage delay time.

3 Addition of Power Model

To more accurately assess the tradeoffs inherent in cache design, we extended the cache model in CACTI to model power consumption.

3.1 Power Estimation

According to [4] and [5], energy consumption can be modeled as

$$E_{DD} = C_L \times V_{DD}^2 \times P_{0\rightarrow 1}$$

where C_L is the physical capacitance of a device and $P_{0\rightarrow 1}$ is the probability that the device will consume energy. We fully account for power dissipation when a capacitor is charged, and ignore discharge events. The energy value obtained from this formula can then be combined with the cycle frequency to provide the dynamic power consumption. For example, a device that consumes 3nJ of power and is clocked at 500MHz will consume 1.5W of power. Our goal with CACTI is to provide the energy consumption in nanojoules, which can then be used to find the dynamic power consumption, depending on the frequency of the cache.

3.2 Automatic Supply Voltage Scaling

CACTI requires the user to specify a technology size as a parameter to the model. Aside from being used to scale the access time reported by CACTI, this parameter scales the capacitances and the value of V_{DD} used by the power model. The value of V_{DD} is scaled by

$$V_{DD} = \frac{4.5V}{(\frac{0.8}{TECH})^{.67}}$$

Where $TECH$ is the feature size of the technology. This means that voltage will scale at a slower rate than capacitance and therefore than access time. The voltage level to which the bitlines are charged is calculated as a fraction of the scaled value of V_{DD}. We allow a maximum V_{DD} of 5V and a minimum value of 1V.

3.3 Power Model

Since the CACTI model tracks the physical capacitance of each stage of the cache model, we use the energy consumption equation from 3.1 to calculate the power consumed at each stage. Additionally, we need to factor in the switching activity and the number of such devices in the cache (as CACTI models the activity down one particular path in the cache).

As an example, consider the power consumption modeled for the decoder on the data path of a set associative cache. CACTI models the decoder as being composed of three stages: the inverter that drives the probe address

bit, the NAND gate that generates the 1-of-8 code, and the NOR gate that combines the 1-of-8 codes and drives the wordline driver (Figure 9 in [6]).

For the first stage, there are $log_2(\frac{C}{B \times A \times Ndbl \times Nspd})$ address bits, and we can estimate that a quarter of these will require the inverter to undergo a $0 \rightarrow 1$ transition (i.e. half of the address bits will be 0's and half of these were 1's before). However, we need both the true and complement forms of the address bits. So the energy consumption of the first stage can be represented as

$$E_{DD_1} = C_{stage1} \times V_{DD}^2 \times 0.25 \times log_2(\frac{C}{B \times A \times Ndbl \times Nspd}) \times 2$$

The next stage is composed of $\lceil \frac{1}{3} log_2(\frac{C}{B \times A \times Ndbl \times Nspd}) \times 2 \rceil$ blocks in each subarray. Each N_{3to8} block is composed of 8 NAND gates. We can estimate that half of these will undergo energy consuming switching. Since there are $Ndbl \times Ndwl$ decoders, the energy consumption is

$$E_{DD_2} = C_{stage2} \times V_{DD}^2 \times Ndbl \times Ndwl \times 4 \times \lceil \frac{1}{3} log_2(\frac{C}{B \times A \times Ndbl \times Nspd}) \rceil$$

Finally, the last stage is composed of the NOR gate that will drive a single wordline. Only one of the NOR gates in the decoder will be selected, which implies

$$E_{DD_3} = C_{stage3} \times V_{DD}^2$$

3.4 Integration of Timing and Power Models

In the original version of CACTI, a cache configuration was chosen that optimized the access time of the cache. To optimize both power consumption and access time, we first generate the maximum values for each measurement over all configurations of a particular set of input parameters. Then, we iterate through the different configurations again, optimizing the following relationship:

$$\frac{access\_time}{maximum\_access\_time} + \frac{power\_consumption}{2 \times maximum\_power\_consumption}$$

We chose to divide the power ratio by a factor of two to emphasize optimization of the access time. This of course could be removed to optimize evenly across both measurements.

3.5 Prior Work

Kamble and Ghose proposed analytical models to estimate energy dissipation in [2]. They looked at using a simulation tool called CAPE which allowed them to track the transistions encountered in different components of the cache. They also investigated a number of architectural power reduction techniques.

4 Results

First, we show a comparison of the results obtained with the new CACTI model to results obtained using SPICE. Then, we present CACTI results for a number of cache configurations and port configurations. The results presented in this section are for the $0.80 \mu m$ technology size and are for caches with 32 byte block sizes, 32 bit addresses, and 64 bit outputs. Unless otherwise specified, the caches in this section have a single read/write port.

4.1 SPICE Verification

As a sanity check, we compare results obtained in CACTI to a cache model implemented in SPICE. Figure 7 compares cache access times in nanoseconds on a logarithmic scale across a variety of cache configurations for both models. Figure 8 compares cache power consumption in nanojoules on a logarithmic scale across the same cache configurations. The models show good correlation, even at larger cache sizes and associativities. These figures show how access time and power consumption grow as cache size increases. The direct mapped cache

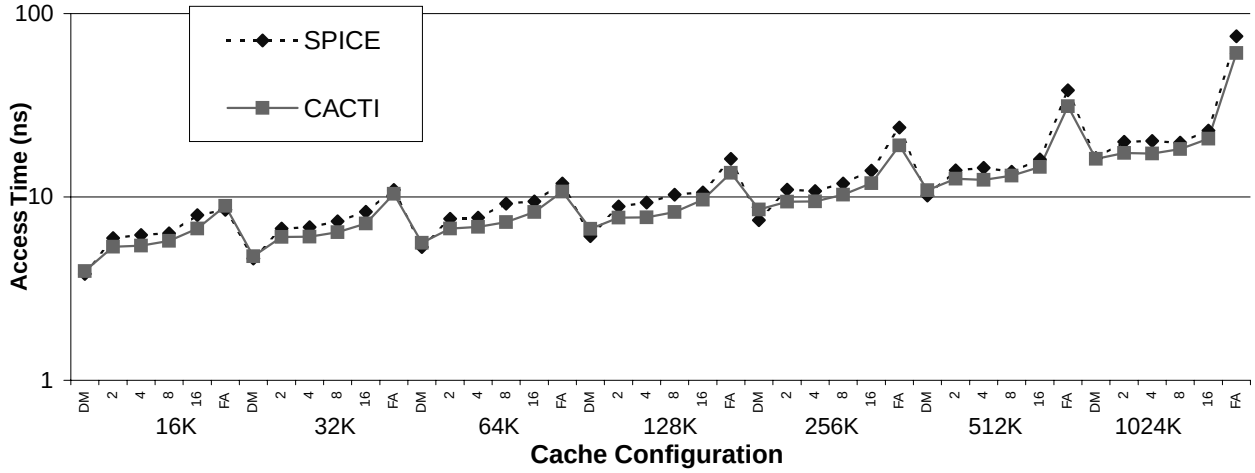

Figure 7: Comparison of access times for a variety of cache configurations in both CACTI and SPICE. The y-axis is on a logarithmic scale, and shows the cache access time in nanoseconds. The x-axis ranges over a variety of cache configurations - all with 32 byte block sizes. Caches sizes of 16K, 32K, 64K, 128K, 256K, 512K, and 1024K are all shown, each broken down into six associativity configurations (direct mapped, 2-way, 4-way, 8-way, 16-way, and fully associative). So the first six points on the graph represent the six different 16K caches, then the next set of six corresponds to the 32K caches, and so forth.

configurations generally consume the least amount of power and can be accessed in the fastest time for each different cache size. By comparison, the fully associative cache takes the longest time to access, but for small cache sizes has lower power consumption than the 16-way set associative case.

4.2 Timing Results

Figure 9 shows access times for the cache configurations we selected. It is interesting to note the similarity in access times between the 2-way and the 4-way associative caches. Increasing the associativity of the cache does increase the number of sense amps that must be used and increases the number of bitlines connected to a single wordline, but it also reduces the number of rows in the decoder. For many cases, the 2-way set associative case proved to perform better with a higher value of Ndbl or Nspd, which effectively reduces the number of rows in the decoder in much the same way as increasing the associativity. However, these also carry the same detrimental effects as increasing the associativity. Unfortunately though, increasing the number of subarrays has an additional consequence - increasing the degree of multiplexing at the bitline column multiplexors. Since we limit the degree of multiplexing that can occur at these multiplexors, the more subarrays there are, the less the column multiplexors can filter output bits from the cache line (Section 2.2.3). This can have a detrimental effect on the performance of the tag path, as it will increase the delay of the comparator and output multiplexors.

This is seen in the 512K cache. Here, the 4-way case performs as well as the 2-way case (12.4ns compared to 12.6ns), despite the increase in associativity. Both have 8 bitline divisions (Ndbl), but the 2-way case has an Nspd of 2 (effectively mapping two sets to a single wordline). This means that they have the same number of rows in their decoders, and effectively the same amount of decoder and wordline delay. However, since we limit the degree of column multiplexing to 16, the 2-way case is unable to filter out the same number of output bits as the 4-way case. But, since the 4-way case has a higher associativity, both cases end up with the same number of sense amplifiers and output drivers. So the data path delay for both is identical. The tag path differs slightly though. The 2-way case has better decoder and tag array performance, at the cost of comparator performance (due to the

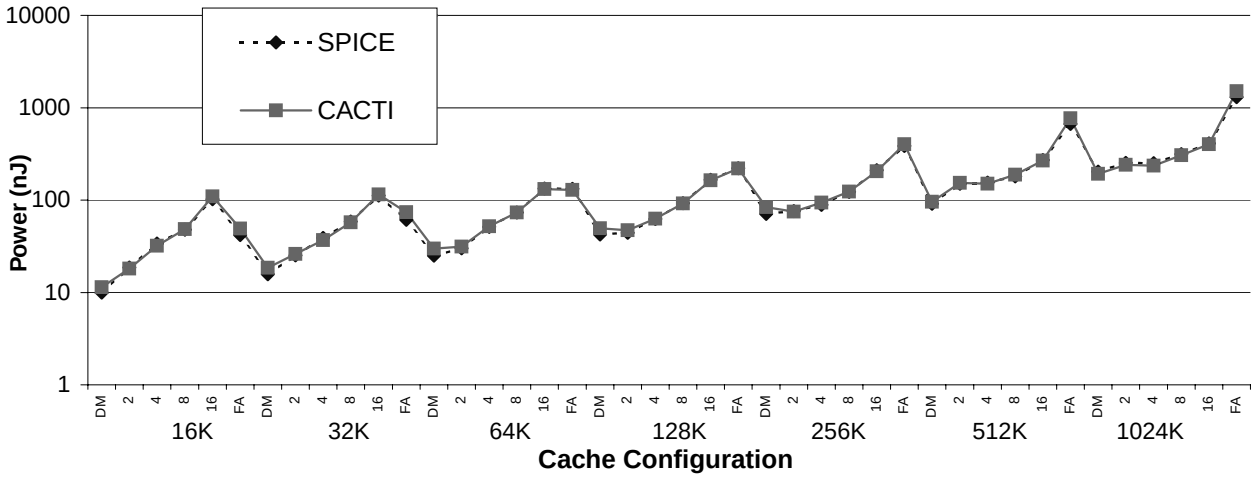

Figure 8: Comparison of power consumption for a variety of cache configurations in both CACTI and SPICE. The y-axis is on a logarithmic scale, and shows the cache power consumption in nanojoules. The x-axis ranges over a variety of cache configurations - all with 32 byte block sizes. Caches sizes of 16K, 32K, 64K, 128K, 256K, 512K, and 1024K are all shown, each broken down into six associativity configurations (direct mapped, 2-way, 4-way, 8-way, 16-way, and fully associative).

increase in Ntspd). Additionally, the extra output bit multiplexing that it must perform increases the delay of the output multiplexors. This causes the 2-way case to have a slightly higher tag path delay than the 4-way case (but only by about 0.2 ns).

4.3 Power Results

Figure 10 shows power consumption for the cache configurations we selected. A majority of power dissipated by the set associative configurations is in the bitlines and sense amplifiers. Therefore, as the number of sense amps required grows (in response to the number of subarrays, the associativity, the value of Nspd, etc) the power consumption also grows. However, for the fully associative configuration, most of the power is consumed in the decode stage (where the tag check is performed). However, the fully associative case does not require a significant number of bitlines to discharge - only enough for a single cache line. This is due to the fact that each wordline only maps to a single cache entry. Therefore, at smaller cache sizes, the fully associative cache uses less power than a highly associative cache (like the 16-way case) as it has substantially less bitline activity. Moreover, a highly set associative will also require more sense amps than the fully associative case. At larger cache sizes, the delay of the fully associative decoder grows considerably and it consumes the most power of any cache associativity configuration we investigated.

Figure 11 shows a breakdown of power consumption for a 64K 2-way associative cache. The data path is the predominant power consumer in this case - 70% is consumed by the data bitlines (40%) and sense amps (30%). There are 128 sense amps in this cache configuration, and 512 pairs of bitlines. There are only 128 rows in the 4 data decoders, and as can be seen in the figure, the data decoder is only responsible for 11% of the total power consumption.

Figure 12 contrasts this, as 84% of the total power consumption is found in the decoder. In a fully associative cache, the decoder also performs the tag check and has a wordline for every entry in the cache. There are 2048 rows in this case - 64 in 32 decoders. Each of these rows has a address comparison line that must be precharged after every unsuccessful tag check. In this model, there are 256 sense amps - but these are still only responsible

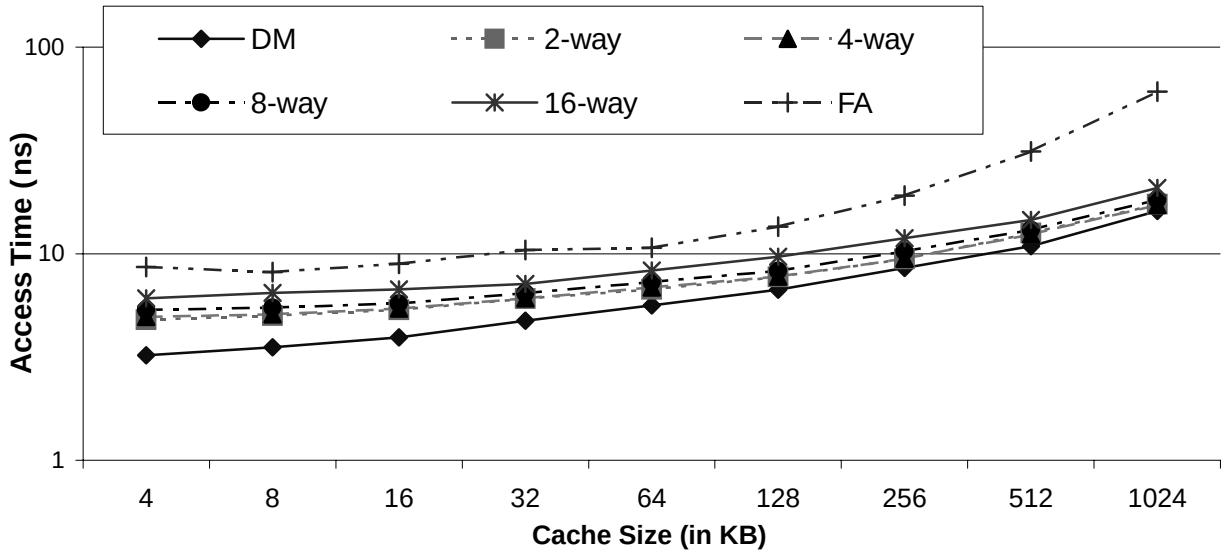

Figure 9: Access times for a variety of cache configurations. The y-axis shows the access time in nanoseconds on a logarithmic scale. The x-axis shows a range of cache sizes in KB. Six lines are plotted, each representing a different kind of associativity.

for 13% of the total power consumption, despite the fact that they are a significant source of power consumption (consuming around twice as much power as the sense amps in the 2-way set associative case, as would be expected). However, there are only 32 pairs of bitlines in this case, and they are only responsible for 2% of the total power consumed.

4.4 Multiported Results

Figure 13 shows the access times for four different port configurations. Figure 14 shows the power consumption for these configurations. We examined a variety of cache configurations using a single ported cache, a single ported cache with an extra read port, a dual ported cache, and a dual ported cache with an extra read port. As can be seen, additional ports lengthen cache access times, especially for large sized caches. An extra read only port provides a slightly smaller increase in access time over an extra read/write port. Moreover, adding extra ports has a tremendous impact on cache power consumption. Because the extra port effectively implies replicating most cache structures and because of the extra physical area involved, the additional power required by a second port is often greater than the total power of a single ported version of the cache.

5 Future Work

In order to completely assess the relative merits of a particular cache configuration, this timing and power model would benefit from the inclusion of an area model. This would be particularly useful in determining the impact of the various cache design parameters (Ndbl, Nspd, etc).

References

[1] C. Thomas Gray, Wentai Liu, and Ralph K. Cain III. *Wave Pipelining: Theory and CMOS Implementation*. Kluwer Academic Publishers, Norwell, MA, 1993.

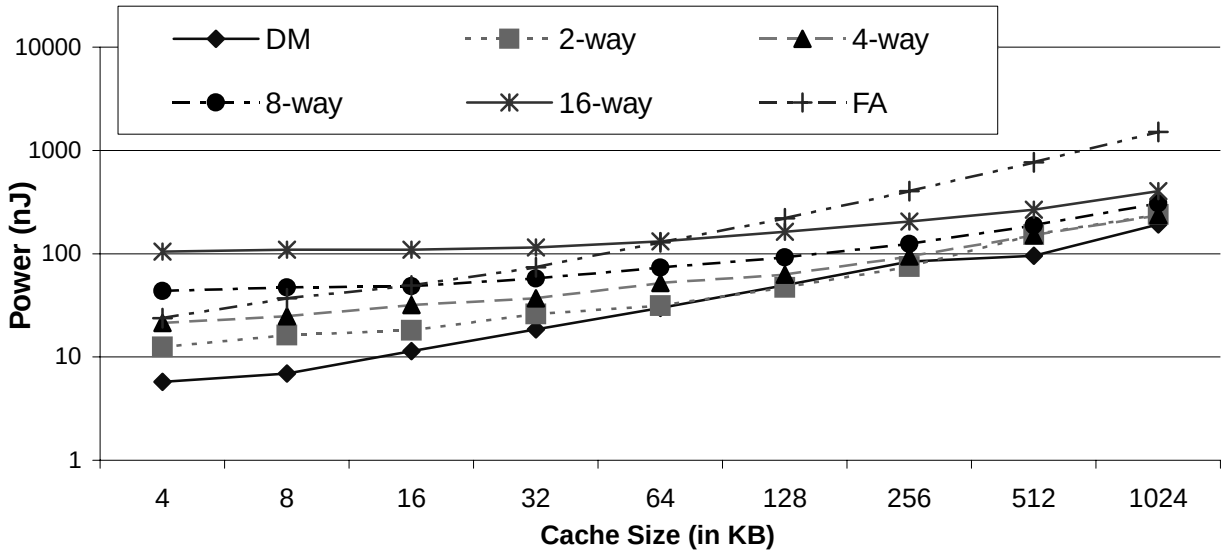

Figure 10: Power consumption for a variety of cache configurations. The y-axis shows the power consumed in nanojoules on a logarithmic scale. The x-axis shows a range of cache sizes in KB. Six lines are plotted, each representing a different kind of associativity.

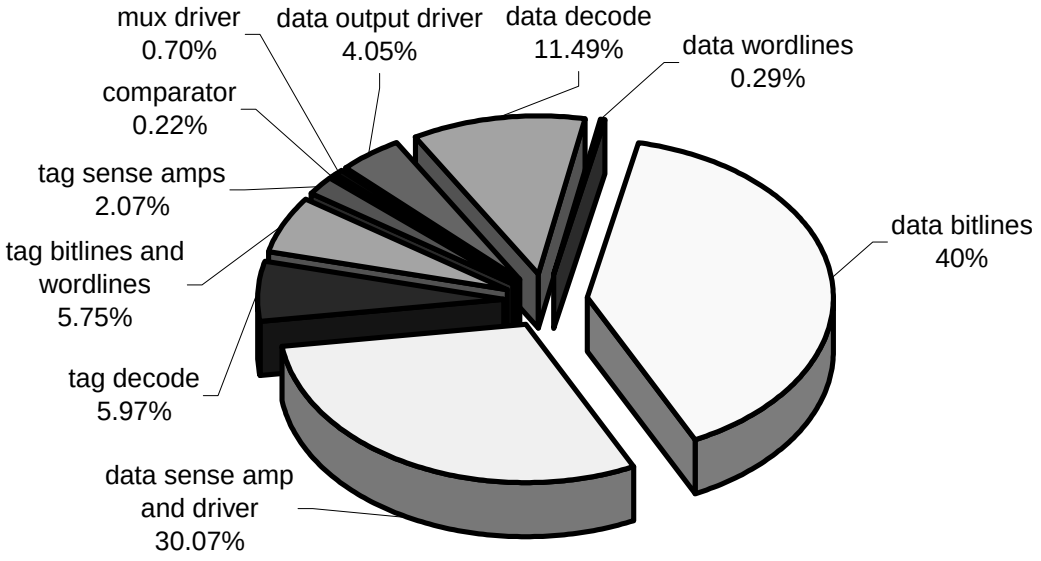

Figure 11: Breakdown of power consumption for a 64K 2-way associative cache. The data bitlines and sense amps are responsible for 40% and 30% of the power consumption of the cache, respectively.

[2] M. Kamble and K. Ghose. Analytical energy dissipation models for low power caches. *International Symposium on Low Power Electronics and Design*, 1997.

[3] Richard E. Kessler. The alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, March/April 1999.

[4] J. Rabaey. *Digital Integrated Circuits*. Prentice Hall, 1996.

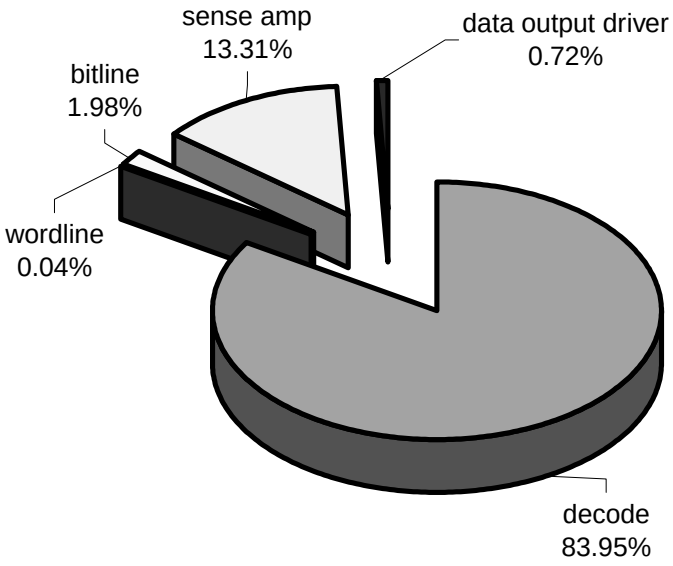

Figure 12: Breakdown of power consumption for a 64K fully associative cache. The decode portion of the cache (including tag comparisons) is responsible for 84% of the total cache power consumption.

[5] N. Weste and K. Eshraghian. *Principles of CMOS VLSI Design*. Addison Wesley, 1993.

[6] S. Wilton and N. Jouppi. Cacti: An enhanced cache access and cycle time model. In *IEEE Journal of Solid-State Circuits*, May 1996.

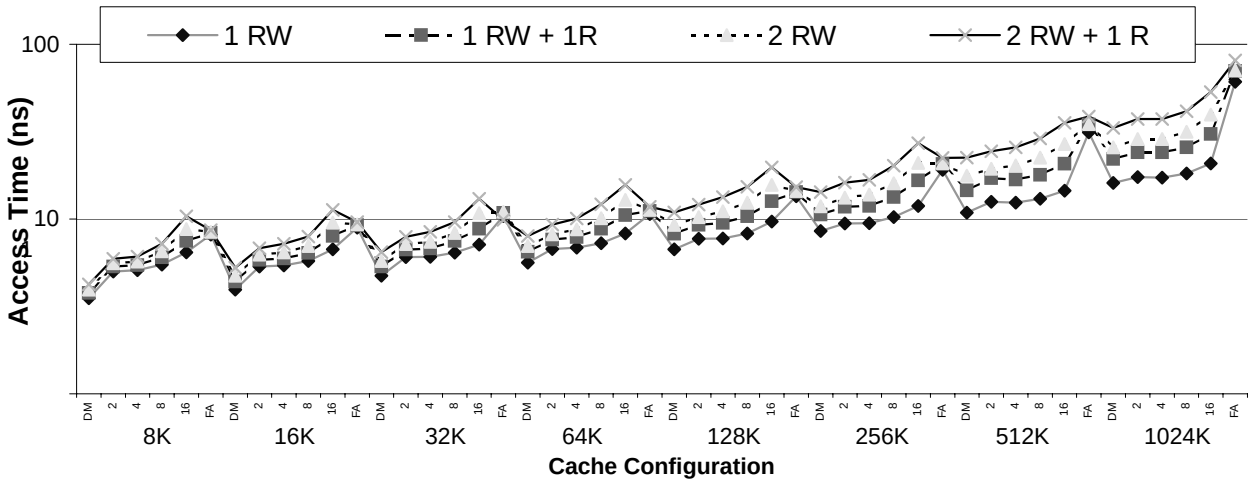

Figure 13: Comparison of access times for a variety of cache configurations. The y-axis is on a logarithmic scale, and shows the cache access time in nanoseconds. The x-axis ranges over a variety of cache configurations. Caches sizes of 16K, 32K, 64K, 128K, 256K, 512K, and 1024K are all shown, each broken down into six associativity configurations (direct mapped, 2-way, 4-way, 8-way, 16-way, and fully associative). Four lines are plotted: a single ported cache, a single ported cache with an extra read port, a dual ported cache, and a dual ported cache with an extra read port.

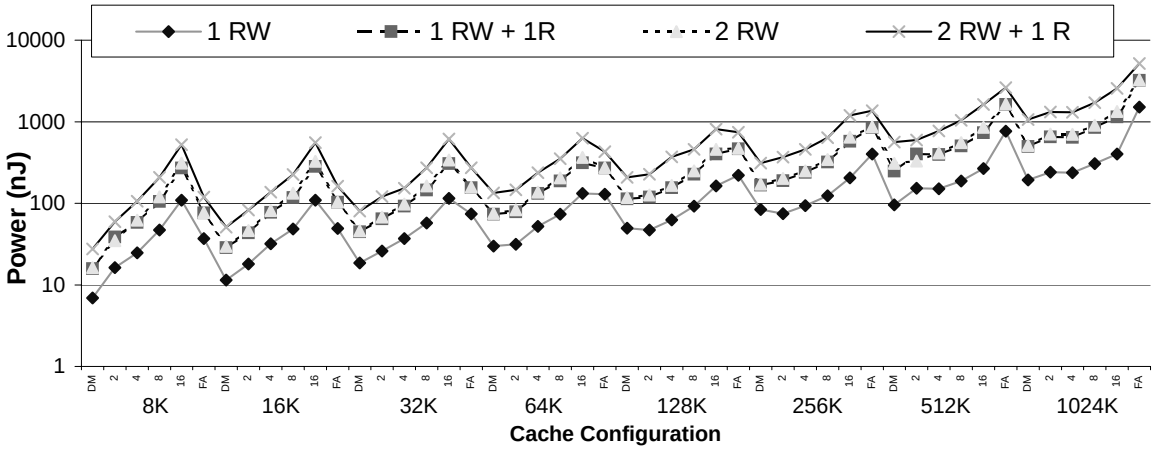

Figure 14: Comparison of power consumption for a variety of cache configurations. The y-axis is on a logarithmic scale, and shows the cache power consumption in nanojoules. The x-axis ranges over a variety of cache configurations. Caches sizes of 16K, 32K, 64K, 128K, 256K, 512K, and 1024K are all shown, each broken down into six associativity configurations (direct mapped, 2-way, 4-way, 8-way, 16-way, and fully associative). Four lines are plotted: a single ported cache, a single ported cache with an extra read port, a dual ported cache, and a dual ported cache with an extra read port.

A CACTI Syntax

```
cacti <csize> <bsize> <assoc> <tech>
OR
cacti <csize> <bsize> <assoc> <tech> <RWP> <RP> <WP>
```

`csize` - size of cache in bytes *(i.e. 16384)*

`bsize` - block size of cache in bytes *(i.e. 32)*

`assoc` - associativity of cache *(i.e. 2 or FA)*

direct mapped caches - *DM*

set associative caches - number *n* (where cache is *n*-way associative)

fully associative caches - *FA*

`tech` - technology size in microns *(i.e. 0.8 or 0.35)*

`RWP` - number of read/write ports *(defaults to 1)*

`RP` - number of read ports *(defaults to 0)*

`WP` - number of write ports *(defaults to 0)*

B CACTI Output

The command
`cacti 16384 32 2 0.80um`
will return the following timing and power analysis for a 16K 2-way set associative cache with a 32-byte blocksize at a 0.80um feature (technology) size:

```
Cache Parameters:
Size in bytes:  16384
Number of sets:  256
Associativity:  2
Block Size (bytes):  32
Read/Write Ports:  1
Read Ports:  0
Write Ports:  0
Technology Size:  0.80um
Vdd:  4.5V

Access Time (ns):  5.34319
Cycle Time (wave pipelined) (ns):  1.78106
Power (nJ): 17.4514
Wire scale from data sense amps to data output:  0.10

Best Ndwl (L1):  1
Best Ndbl (L1):  2
Best Nspd (L1):  1
Best Ntwl (L1):  1
Best Ntbl (L1):  2
Best Ntspd (L1):  2
Nor inputs (data):  3
Nor inputs (tag):  2

Time Components:
data side (with Output driver) (ns):  4.6761
tag side (with Output driver) (ns):  5.34319
decode_data (ns):  1.39527
(nJ): 0.679041
wordline and bitline data (ns):  1.64209
wordline power (nJ): 0.0919375
bitline power (nJ): 6.71302
sense_amp_data (ns):  0.58
(nJ): 4.27455
senseext_driver (ns):  0.412297
(nJ): 2.73009
decode_tag (ns):  1.06472
(nJ): 0.410481
wordline and bitline tag (ns):  0.641881
```

```
wordline power (nJ): 0.0182812
bitline power (nJ): 0.621324
sense_amp_tag (ns):  0.26
(nJ): 0.562801
compare (ns):  1.29042
(nJ): 0.0744832
mux driver (ns):  1.24989
(nJ): 0.215418
sel inverter (ns):  0.189851
(nJ): 0.0023513
data output driver (ns):  0.646435
(nJ): 1.05761
total data path (without output driver) (ns):  4.02966
total tag path is set assoc (ns):  4.69676
```

C Code

C.1 CACTI

CACTI contains the following files/directories:

- *main.c* - main module
- *io.c* - input/output
- *time.c* - model implementation
- *def.h* - technology specific information, transistor widths, threshold voltages
- *makefile* - compilation information and predefined runs
- *data/* - holds completed CACTI runs
- *spicemodel/* - contains the spice cache model

The makefile script can be used to run a variety of cache configurations, and store the results in the `data` directory. The syntax of a CACTI run name is:

$$\text{cache}C\text{-}A\text{-}FS\text{um-}RWP\text{-}ERP\text{-}EWP$$

where:

C = cache size in KB

A = associativity

FS = technology size in microns

RWP = read/write ports

ERP = extra read-only ports

EWP = extra write-only ports

Many preconfigured runs are stored in the makefile. To run one of these, just do `make` *runname* (i.e. `make cache16-2-0.80um-1-1-0` or `make cache32-FA-0.35um-1-0-0`). A CACTI run produces two files - both with the *runname* prefix and stores them in the data/ directory. *runname.out* is the CACTI timing and cache configuration output. *runname.aux* is a small header file that can be used in conjunction with the spice cache model. It contains configuration information discovered by CACTI (i.e. *Ndbl*, *Nspd*, etc). To generate all preconfigured runs used in this study, use *make runall*.

C.2 Spice

The spice directory (spicemodel) in the CACTI main directory contains the following files

- *cache.prehsp* - the cache model template for spice
- *widths* - transistor widths and other tech data
- *Makefile* - compilation information and predefined runs
- *data/* - holds completed spice run output
- *otherdata/* - holds completed spice run data (for use with awaves)

The spice model Makefile uses the same conventions as the cacti model. It copies the .aux files from the CACTI directory and uses them to guide the run.