Design Automation for Embedded Systems, 7, 35–51, 2002.

Tuning of Cache Ways and Voltage for Low-Energy Embedded System Platforms*

TONY GIVARGIS givargis@ics.uci.edu
Center for Embedded Computer Systems, Department of Information & Computer Science, University of California, Irvine, CA 92697

FRANK VAHID vahid@cs.ucr.edu
Department of Computer Science & Engineering, University of California, Riverside, CA 92521. Also with the Center for Embedded Computer Systems at UC Irvine

Abstract. System-on-a-chip platform manufacturers are increasingly adding configurable features that provide power and performance flexibility, in order to increase a platform's applicability to a variety of embedded computing systems. We illustrate the energy benefits of combining the configurable features of voltage scaling and cache way shutdown in a single platform. We describe methods to assist a designer to tune such a platform to a particular software task and to particular energy optimization criteria.

Keywords: Adaptive architectures, embedded systems, low energy, low power, platform tuning, real-time systems, system-level exploration, system-on-a-chip.

1. Introduction

System-on-a-chip (SOC) platforms are becoming popular as a means of reducing time-to-market. A platform is a pre-designed computing system, typically consisting of a microprocessor, memory hierarchy, coprocessors, peripherals, and possibly field-programmable logic. An IP (Intellectual Property) platform comes in the form of a hardware description language. An IC (Integrated Circuit) platform comes in the form of a chip. An IC platform can be oriented towards prototyping, or can be oriented towards implementation in a final product. In this paper, we focus on product-oriented IC platforms with configurable parameters.

Product-oriented IC platforms become cheaper as they are produced in higher volumes. Thus, platform manufacturers seek to make a platform as general and hence widely applicable as possible, without allowing that generality to increase platform size too much to offset the volume cost savings. One increasingly common method for making a product widely applicable is to make its voltage source scalable [4], [7], [9], [14], [15]. A platform user (i.e., an embedded system designer) can then set the voltage level higher for fast performance, or lower for reduced power, depending on the end application's static or dynamic requirements.

* This work was supported by an NSF CAREER award (CCR-9876006) and TriMedia Technologies Inc.

Another method for making a platform widely applicable is to include an N-way set-associative cache whose ways can optionally be individually shut down. Such shut down can reduce power per access by reducing power-costly tag comparisons and eliminating the power necessary to drive a way's bit-lines and word-lines, at the potential expense of reduced performance due to more misses. If this reduction is greater than the increase caused by more cache misses and hence power-costly accesses to the next level of memory, then overall power is reduced. When power was not an issue, shutting down ways was not considered since it could only hurt performance. Motorola's recent version of an MCORE processor IC has a configurable 4-way set-associative unified cache [12], in which each way can be disabled, or used for instructions, data, or both. They show that the best configuration depends heavily on the particular running task. Albonesi [1] proposes a 4-way data cache with ways that can be disabled even during task runtime. They achieve 40% cache energy savings at a cost of only 2% performance for several examples. Other types of cache configurability have also been proposed [2], [17]. Furthermore, various cache power-reduction methods could easily be adapted for use in a configurable cache. For example, pseudo-set-associative cache [11] reduces access power at the expense of some performance loss, by initially searching for a match in the first way of a cache, and only searching the remaining ways if the first way misses. Introducing such a search strategy as a configurable feature to a cache is straightforward.

Both voltage scaling and configurable caches have been shown to provide excellent flexibility for trading off powerand performance. The next logical step is for a platform to have both a scalable voltage and a configurable cache. The problem then arises of *tuning*—configuring of voltage *and* cache parameters to a particular software task and to particular power and performance requirements. In the case of a platform executing a single task, a single configuration may be established during system initialization. In the case of multiple tasks in a real-time system, each task may cause a reconfiguration of the platform.

In this paper, we describe an example platform having configurable voltage and cache, and we show the extensive power and performance tradeoffs possible through tuning of such a platform. We discuss methods for a platform developer to support tuning of the platform. We describe methods for tuning such a platform to a task under different optimization criteria, such as minimizing energy, and minimizing energy-delay.

2. Evaluating Configurable Architectural Features

2.1. Architecture

The platform we modeled is shown in Figure 1. It consists of a MIPS microprocessor core, a unified instruction/data cache, an on-chip memory, and a DC-to-DC converter. The microprocessor configures the DC-to-DC converter by setting a register, to scale the input supply voltage from 3.8 V down to 1.0 V, in increments of 0.1 V; the clock frequency is reduced accordingly for lower voltages. The cache is unified and 8-way set-associative, and can be configured by the microprocessor to use 1, 2, 4, or 8 of its ways. We used a 16-byte

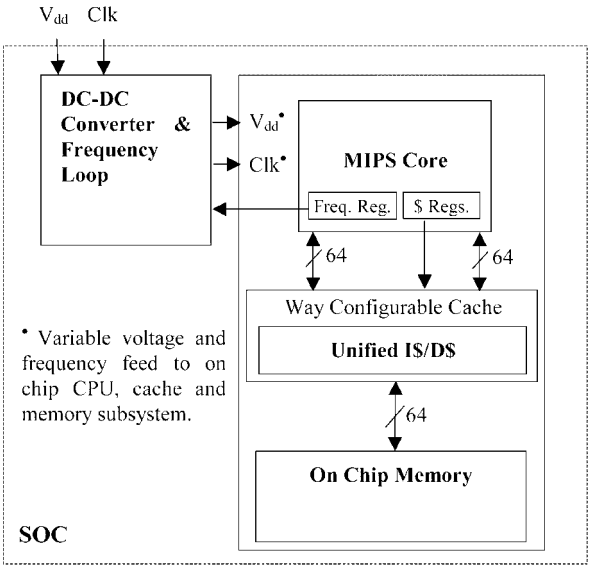

Figure 1. Target architecture.

line size and 64 rows. In [12], the 4-way cache could have each way configured to be disabled, used for instruction-only, for data-only, or for both instruction and data. For simplicity in this work, our cache's ways could only be disabled or used for both instruction and data, but the methods can be applied to cache's with greater or lesser configurability.

2.2. Evaluation Environment

A platform developer must have some way to evaluate the power and performance impacts of a platform's various configurable features, in order to ensure that including such features is beneficial. Furthermore, that method of evaluation may become part of a tuning environment provided to a platform user. One method of evaluation is to measure power and performance of an application executing on a physical instance of a configurable platform. This method is generally preferred when available, due to the speed and accuracy of evaluation. When not possible, due to the unavailability of the platform, or the difficulty or danger of executing the platform in a real environment, then simulation based methods may be preferred.

We utilized a simulation-based method for this work. Our evaluation environment, called Platune, includes an executable model of the target architecture depicted in Figure 1. The simulation model is augmented with power models to allow for measur-

ing the average power consumption of the chip while running a task. Platune can be broken down into two components, namely, the simulator module and power analyzer module. A lengthier description of those components can be found in [8], while a brief summary is given here.

Platune is a tightly coupled collection of event driven cycle accurate simulation models of its various components, namely, processor, cache, memory, and busses. The processor simulator maintains detailed statistics on its internal activity, e.g., fetches, stalls, instruction execution frequency, register file access, floating-point activity, etc. Such statistics are used in a post simulation analysis to compute power and performance metrics. The level of detail of the processor simulator can be compared to the most detailed simulator that is part of the collection of simulators making up SimpleScalar [5]. The cache simulator of Platune is a fully parameterized element that operates on a stream of memory references that is output from the processor. In addition to the standard cache metrics, such as number of misses (e.g., all those generated by Dinero cache simulator [6]), the Platune cache simulator maintains additional activity statistics, including the number of tag comparisons, the word-line activity and bit-line activity, etc., that are later used for power computation. Like the cache simulator, the bus simulator in Platune also operates on a stream of data and memory references that are generated by the processor, cache and memory modules, and accumulates bus wire bit toggle statistics that are later used for power computation.

The second component of Platune, the power model and analyzers, operate on the statistics that are gathered during simulation, as described earlier. For the processor, an instruction based power modeling is applied that is based on models developed in [16] and [3]. For caches, first a structured (physical) model is deduced based on the cache parameter settings and technology feature size. This allows estimation of bit-line, word-line, comparator, storage transistors, and address decoding logic capacitive loads. Then, switching activity from the simulation phase is applied to obtain average power consumption of the cache. Similarly, for each bus segment, a rough layout is inferred that is based on the chip technology, chip area, bus widths, and relative size of the various cores, in order to obtain the average bus capacitance. Then, switching activity from the simulation phase is applied to obtain average power consumption of various buses.

The entire Platune environment is integrated into a single GUI application. The environment bundles in a public-domain C compiler as well as a small runtime kernel for use by the tasks that are being executed on the platform. Overall accuracy of Platune was experimentally shown to be 5% to 15% of gate-level measurements [8].

We assume the evaluation environment provides a procedure *EvaluatePlatformConfig(V, A)*, which can be used by our algorithms. The procedure executes a given task on our example platform using the voltage V and associativity A values passed as parameters, and returns the summary of the evaluation in a data structure *Eval*, which includes the time, power, energy, and voltage and associativity settings. The procedure may fire off simulation tools, like those in Platune, SimpleScalar, hardware description language simulators, or other simulation approaches. It may call higher-level estimators, such as described in [13]. It may even run the task on real hardware if available. The techniques in the rest of the paper can be used with any of these evaluation approaches.

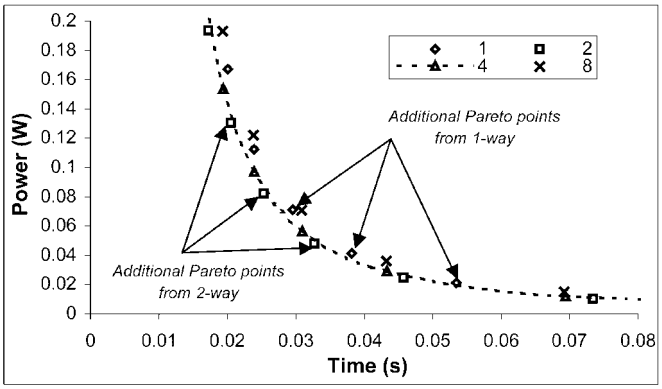

Figure 2. Power vs. time for the *g*3*fax* example. The configurable associativity yields additional Pareto points.

2.3. Power/Performance Tradeoffs and Pareto Points

We now demonstrate the power and performance ranges that are possible through the configurable platform described above. We used three benchmark tasks from the PowerStone benchmark suite [12]. *g*3*fax* is a group three fax decoder (single level image decompression), *jpeg* is a 24-bit JPEG image decompression standard, and *v*42 performs modem encoding/decoding. *g*3*fax* is roughly 652 lines of C code, *jpeg* 620 lines, and *v*42 743 lines.

For each benchmark, we evaluated power and performance for 29 voltage levels ranging from 1.0 to 3.8 in 0.1 V increments, and all 4 cache way configurations, for a total of 116 configurations. We executed 1,127,387, 4,594,120, and 2,441,985 instructions for the *g*3*fax*, *jpeg*, and *v*42 examples, respectively, requiring 512, 2088 and 1110 seconds for each configuration on Platune.

We plot the data for the *g*3*fax* example in Figure 2. The plot uses different symbols for 1, 2, 4, and 8-way associativity—note that, for a given associativity, the upper-left points correspond to the higher voltage levels. The maximum values of the axes are limited to improve viewing of the interesting points; thus, not all points appear.

The configurability of the platform provides a tremendous range of power and performance, mostly due to the voltage scaling. Table 1 summarizes the power (in Watts) and time (in seconds) ranges for each example.

The most important feature from the plot to note is that, while voltage certainly has a larger impact on power and performance, the configurable cache improves the granularity of possible tradeoffs by providing additional Pareto points. A Pareto point is one for which no other point has both better power and time. In other words, Pareto points form the lower-left curve of the plot, and represent the design points of interest. For example, observe the indicated points in Figure 2. The points for a 4-way cache, shown as triangles, are connected by a dashed line. Notice that a 2-way cache's points, shown as squares, represent several additional Pareto points. Likewise, a 1-way cache's points, shown as diamonds, represent even more Pareto points. Thus, if we had a fixed 4-way cache, we would not have been able to achieve the intermediate points achieved by a 2-way and 1-way cache. Similar

Table 1. Power and Performance Ranges for the Examples

Example	Metric	Max	Min	Ratio
g3fax	power	0.2868	0.000195	1471
	time	1.19	0.01625	73
jpeg	power	0.3089	0.000206	1500
	time	5.834	0.07124	82
v42	power	0.2904	0.000164	1771
	time	4.97	0.03862	129

Table 2. Configurable Cache Increases the Pareto Points and Hence the Power/Performance Tradeoff Granularity

Example	With Fixed 4-way Cache	With Configurable Cache
g3fax	29	67
jpeg	29	57
v42	29	50

observations can be made in other regions of the data. We observed similar plots for the other two examples.

That configurable cache leads to additional Pareto points is not an obvious situation—it could have occurred that alternative cache configurations may not have yielded interesting points. For example, notice that an 8-way cache (shown as x's in the figure) does not provide any Pareto points in this example.

Table 2 compares the total number of Pareto points obtained for each example with a configurable cache, to the number obtainable using a fixed 4-way associative cache. The importance of these additional Pareto points will become quite clear in subsequent sections. One import feature of Pareto points is that a minimum energy solution, for any given time constraint, will always be a Pareto point.

To summarize, a platform with both configurable voltage and cache provides for more tradeoffs between power and performance.

3. Developing a Tuning Environment

Presently, tuning a platform's configurable features is left to the platform user. Because the potential configuration space may be very large, platform developers may provide automated support for tuning. The key to such support is to find the best configuration for a particular application without evaluating all possible configurations.

To do this, a platform developer may exhaustively evaluate a number of benchmarks, and find trends in the generated data that can be used to develop algorithms that efficiently

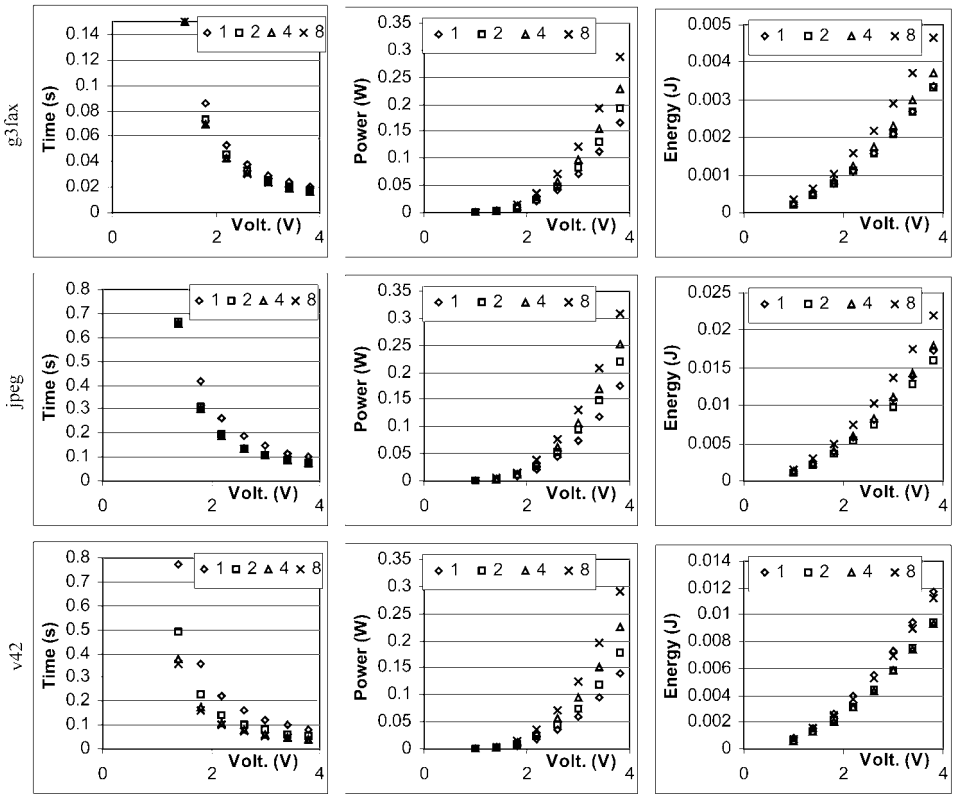

Figure 3. Time, power and energy data for the three examples.

explore the configuration space for typical tuning criteria desired by a user. We consider the typical tuning criteria of minimizing energy given some performance constraint, and of optimizing energy × delay (a.k.a., energy–delay) given a performance constraint, while commenting on other tuning criteria.

3.1. Optimizing Energy Given Timing Constraints

3.1.1. Problem Overview

A common goal is to minimize the energy required to perform a repetitive task. Energy minimization can maximize battery lifetime.

Figure 3 provides exhaustive energy data for all three examples. The *x*-axis represents the voltage level, and different symbols represent different associativity. The figure also

Table 3. Close-up Look at Time, Power and Energy Data for the Lowest Voltage for Our Three Examples

Example	Metric	1-way	2-way	4-way	8-way
g3fax	time	1.19	1.02	0.9643	0.962
	power	0.0001954	0.0002264	0.0002674	0.0003355
	energy	0.0002326	0.000231	0.0002579	0.0003227
jpeg	time	5.834	4.275	4.219	4.218
	power	0.0002059	0.0002581	0.0002938	0.0003613
v42	time	4.97	3.141	2.435	2.287
	power	0,0001637	0.00002077	0.000265	0.0003396
	energy	0.0008138	0.0006522	0.0006452	0.0007766

shows power and time data. We can see that, for our platform, minimum energy comes from setting the voltage as low as possible, and then picking the associativity that best matches the task. For the minimum voltage of 1.0 V in each example, we get the time, power and energy values shown in Table 3.

This data shows that 2-way associativity is best for *g*3*fax* and *jpeg*, while 4-way is best for *v*42.

However, note that the lowest energy configuration may exhibit very poor performance. For example, the *jpeg* example would require 4.275 seconds to decode an image at 1.0 V—too long for most applications. In fact, we had to omit the 1.0 V points from the time plots in Figure 3, as they were literally off the chart.

For reasons obvious from these examples, we consider instead the problem of *minimizing energy while also meeting a performance constraint.* We assume the constraint is given as a maximum time for the task to execute once.

3.1.2. Approximate Overview

Let us begin by developing an approximate algorithm that works well on the generated data, and will form part of an exact algorithm. Looking at the energy plots, we see that lower voltage will usually give us lower energy, independent of associativity. This is not always true—notice that there is some overlap of energy values for adjacent voltage levels. However, minimizing voltage should suffice in our approximate algorithm. For a given voltage, we would then want to find the associativity that yields minimum energy for that voltage.

Note from the time plots that 8-way associative cache always yields the best time at a given voltage. This is logical; shutting down ways can only increase the number of misses and thus lengthen time. Thus, our algorithm can begin by finding the minimum *feasible voltage*—namely, that voltage at which an 8-way cache satisfies the time constraint. No lower voltage could possibly satisfy the time constraint. Given the lowest feasible voltage, the algorithm can try 1, 2 and 4-way caches, in addition to the already-tried 8-way cache, at

```
Eval TuneEnergyApprox( )
// Find lowest feasible voltage that can satisfy
// the timing constraint, using binary search.
Eval eval_init, eval_best, eval;
float V_low;

eval_init = BinarySearchVoltRange (1.0, 3.8, 8, T);
V_low = eval_init.V;

// Find lowest energy associativity at this voltage
// that still satisfies the timing constraint
eval_best = eval_init;
for each associativity A in (4, 2, 1) loop
  eval = EvaluatePlatformConfig(V_low, A);
  if (eval.time > T)
    exit loop; // smaller assoc. would be slower
  if (eval.energy < eval_best.energy)
    eval_best = eval;
end loop;

// eval_best is the lowest energy configuration that
// satisfies the timing constraint at the lowest
// feasible voltage
return(eval_best);
```

Figure 4. Approximate algorithm for minimizing energy while satisfying a performance constraint.

that voltage too, to see which associativity gives minimum energy at that voltage while still meeting the timing constraint. That voltage and associativity would be the best configuration found by the algorithm.

We want to find the lowest feasible voltage in an efficient manner. We could begin with the lowest voltage, evaluate performance, and then increase voltage by the smallest possible voltage step and re-evaluate, until we find the first voltage that satisfies the constraint. Conversely, we could begin with the highest voltage and work down until we fail to meet the constraint. Either approach has a time complexity linear with respect to the number of possible voltage steps. In case the number of possible steps is large (such as 128 [4]), this approach may take too long. However, observe that task runtime strictly decreases as we increase voltage. Thus, we can view the problem of finding the lowest feasible voltage as a problem of searching for an item in an *ordered* list. A fast search method for such a problem is binary search. Thus, we can step through the voltage levels using a binary search approach. We start at the middle voltage level (and an 8-way cache) and evaluate—if this configuration meets the timing constraint, we perform binary search on the range of lower remaining voltages. If it doesn't, we perform binary search the higher remaining voltages.

Our approximate algorithm, *TuneEnergyApprox*, is shown in Figure 4. It uses the *EvaluatePlatformConfig* routine described in Section 2.2. The algorithm begins by calling a subroutine *BinarySearchVoltRange*, which takes as parameters the lowest and highest voltage settings to examine, the cache associativity, and the time constraint T. It steps through voltage settings in a binary search manner, as described above, and makes calls

to *EvaluatePlatformConfig* for each setting, returning the evaluation with the lowest feasible voltage, which is stored in a variable *V_low*. Then, the algorithm enters a loop that evaluates associativities lower than 8, starting with 4, then 2, then 1, at the voltage *V_low*. If a lower associativity fails to meet the timing constraint, the loop terminates, since even lower associativities would also fail. At the end of the loop, *eval_best* will be the evaluation having the lowest energy encountered, and so it is returned by the algorithm.

The algorithm has a worst-case time complexity of $\log_2(Vn) + An$, where Vn is the number of possible voltage settings, and An is the number of possible associativities. For our platform, Vn is 29 and An is 4, so the worst-case number of evaluations is $5 + 4 = 9$. This can be compared with an exhaustive search of all configurations, having time complexity of $29 \times 4 = 116$.

As an example, consider the time plot for *g3fax*, and suppose we are given a time constraint of 0.6 seconds (shown as a dashed line in Figure 3). We see that a voltage setting of 2.2 V, with any of the cache configurations, would satisfy this time constraint, whereas 1.8 V would be too slow. Given that we have selected a voltage of 2.2 V, we want to pick the associativity that gives the best energy. Looking closely at the energy data reveals that 2-way associativity yields less energy at any voltage level. Thus, we could select a configuration of 2.2 V and 2-way associativity.

Note that 2-way associativity provides neither the best time nor the best power at 2.2 V. Best time will usually be obtained by activating the entire cache, in this case, by using 8-ways. Best power, in cases where cache power dominates overall power, will be obtained by shutting down most of the cache, in this case, by using 1-way. Thus, after determining what voltage level will satisfy timing, we must then look directly at energy.

As another example, consider the time plot for *v42*, and a time constraint of 0.2 seconds (shown as a dashed line in Figure 3). A voltage of 1.8 V with 8-way associativity yields a time of 0.16 seconds, thus satisfying the time constraint. Reducing associativity to 4-way yields a time of 0.18 seconds, still satisfying the constraint. However, reducing associativity to 2-way yields a time of 0.23 seconds, which exceeds the constraint. Thus, our algorithm compares energy values for 4-way and 8-way associativies at 1.8V, and seeing that 4-way is better, returns a configuration of 1.8 V and 4-way associativity.

3.1.3. Exact Algorithm

The above algorithm is approximate, because it does not consider configurations using higher voltage than the minimum feasible voltage. However, for a particular example, a time-satisfying configuration could exist that uses a higher voltage and a different cache configuration but has lower energy than the best time-satisfying configuration at a lower voltage level. This situation can be seen in the energy plots of Figure 3. At the right side of the *g3fax* and *jpeg* plots, for example, we see that 2-way associativity at one voltage level has lower energy than 8-way associativity at the next lower voltage level. Assume for the moment that our cache were only configurable as 2 or 8 way, and consider the *g3fax* time plot in Figure 3. Though it is hard to see from the plot, at 3.0 V, 2-way has a time of 0.0253 seconds, while 8-way has a time of 0.02385 seconds. Now, suppose the time constraint was

```
Eval TuneEnergyExact ()

// Find approximate solution
Eval eval_best, eval;
eval_best = TuneEnergyApprox( );

// Create too-slow list of all associativities less
// than eval_best.A
LIST tooslowlist, newlist;
for (int A = (eval_best.A)/2; A >= 1; A = A/2) loop
  tooslowlist.Append(eval_best.A);
end loop;

// Look for higher-voltage assocs. with lower energy
float V = eval_best.V;
while (((V = NextHigherVoltage(V)) != NONE) and
     (!tooslowlist.IsEmpty( ))) loop
  // Find new assocs. that satisfy time constraint
  for each A in tooslowlist loop
    eval = EvaluatePlatformConfig(V, A);
    if (eval.energy > eval_best.energy)
      tooslowlist.Remove(A); // already worse
    else if (eval.time <= eval_best.T)
      newlist.Append(A);
      tooslowlist.Remove(A); // gets one chance
    else // eval.time > eval_best.T
      exit loop; // smaller assocs. can't satisfy
  end loop;

  // Check if new assocs. reduce energy
  for each A in newlist loop
    if (eval.energy < eval best.energy)
      eval best = eval;
  end loop;
end loop;

return(eval_best);
```

Figure 5. Exact algorithm for minimizing energy while satisfying a performance constraint.

0.024 seconds. The lowest feasible voltage would be 3.0 V, and the best cache configuration at this voltage would be 8-way, so our algorithm would find this point, having energy 0.002905 J. However, note from the *g3fax* energy plot that a configuration of 3.4 V and 2-way cache actually exhibits lower energy—0.002671 J. Our approximate algorithm would not detect this point, since it would never check voltages higher than 3.0 V.

We provide an exact algorithm in Figure 5. The algorithm begins by calling the approximate algorithm to find the lowest feasible voltage and the smallest time-satisfying associativity at that voltage. We refer to the smaller associativities at this voltage level that exceed the time constraint as the *too-slow* list. The algorithm then improves on the approximate

algorithm—as long as there are higher voltages to examine and the too-slow list is not empty, the algorithm iterates as follows.

The algorithm increases the voltage to the next higher level, and then calls *EvaluatePlatformConfig* with each associativity, highest to lowest, moving to the next step when an associativity doesn't meet the timing constraint. If the evaluated energy for this associativity is greater than the lowest so far, this associativity is removed from the too-slow list and not considered further—its energy will only get worse at higher voltages, according to the trends in Figure 3. Otherwise, if the evaluated time now meets the time constraint, this associativity is removed from the too-slow list and is added to a new list. The algorithm then checks the energy for the associativities in the new list, and updates the best configuration seen so far.

In summary, the idea is to increase to the next higher voltage and catch any smaller associativities that didn't satisfy timing at the lower voltage, but do satisfy timing at this higher voltage and happen to have lower energy than the best configuration seen so far.

The approximate part of the algorithm again has worst-case time complexity of $\log_2(Vn) + An$. The subsequent iterations in the worst-case may step through Vn voltages in the exact algorithm's while loop. The two for loops in that while loop have an amortized iteration count of O(1) (each item in the too-slow list will be removed only once and then checked for energy only once). So the complexity of the exact algorithm is $\log_2(Vn) + An + Vn$. However, we rarely expect to see this worst case, since we don't expect to have to increase voltage by much before other associativities have higher energy than the best seen so far, thus causing the too-short list to empty quickly.

Further improvements to the algorithm involve increasing the voltage during the iterative step in larger chunks, backtracking when necessary—one can approach binary search in this direction too, thus reducing the Vn term closer to $\log_2(Vn)$.

Recall that we consider only a basic form of configurable cache. Other caches may be more highly configurable. For example, there may be more ways (e.g., a 32-way cache), and any number of ways could be shut down rather than just powers of two (e.g., 7 ways may be active). Ways could be configurable for instructions, data or both. Lookup could be configurable as set-associative or pseudo-set-associative. Both our approximate and exact algorithms can be extended for more-configurable caches. The key is to sort the configurations by their time impact, and to replace occurrences of lists of associativities in our algorithms by the more general idea of lists of cache configurations. If certain configurations can't be sorted relative to one another, they must be treated as a set within the list. Furthermore, if energy is not strictly increasing at higher voltages for a given configuration, then we would simply remove the line with the comment "already worse" in the exact algorithm.

3.2. Optimizing for Energy × Time

As mentioned earlier, optimizing for energy only may result in unacceptably slow performance. One way to solve this is to provide a time constraint. However, if a hard time constraint does not exist, an alternative that has been proposed by other authors is to optimize the product of energy and time [10]. Optimizing this product seeks to balance the

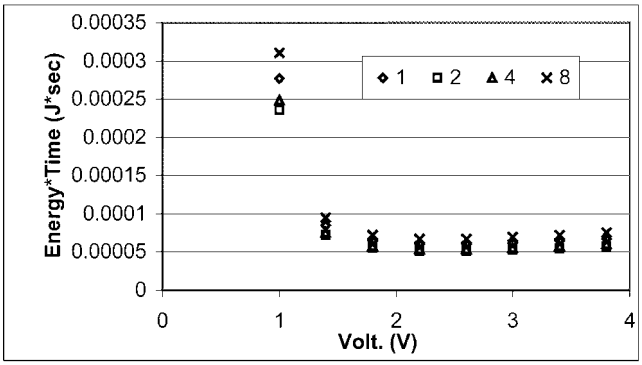

Figure 6. Energy × time for the *g3fax* example.

minimization of energy with the minimization of time, not letting one grow unacceptably large.

We provide energy × time data in Figure 6 for *g3fax*. Notice that there is a clear minimum point at 2.2 V. All three examples exhibited a similar curve with a clear minimum, though at different voltage levels. Furthermore, note that the ordering of associativities is the same at each voltage level—this was also consistent across examples. Given these trends, we can develop a straightforward exact algorithm, similar to the approximate algorithm in Figure 4. To find the voltage corresponding to the minimum energy × time, we perform a modified binary search of the voltage. The modification to binary search is as follows. Given a voltage range, we find the two middle voltages of the range and evaluate them. If the slope of the line connecting the resulting energy × time points is positive, then we must be to the right of the minimum, and thus we consider only the remaining voltage range less than the middle voltages. If the slope is negative, we must be to the left, and thus we consider the remaining range greater than the middle voltages. After finding the voltage corresponding to minimum energy × time, we evaluate all possible associativities at this voltage, and choose the one with minimum energy × time. This algorithm is exact, and has worst-case time complexity $\log_2(Vn) + An$.

3.3. Optimizing for Other Situations

There are several other tuning scenarios that can be solved using slight variations to the above algorithms.

A platform user might, for example, have a time constraint, but want to minimize energy rather than power. This can be solved using a similar approximate algorithm or exact algorithm as proposed for minimizing energy, substituting power evaluations for energy evaluations.

A platform user might instead have a power constraint, and wish to minimize time. Again, we can use similar techniques. We can use an approximate algorithm that performs binary search on the voltage range to find the maximum feasible voltage that satisfies the power constraint, and then pick the best associativity. We can extend this into an exact algorithm by iterating for lower voltages, examining the remaining set of associativities at each stage.

Several other possible variations to the techniques exist for other optimization goals.

The specific algorithms may differ for different platforms—the key point is that the platform developer can analyze extensive data to generate fast tuning algorithms, which may or may not look similar to those presented above. In some cases, exact algorithms may be too time consuming, so approximate algorithms might be all that is possible.

3.4. Dependency on Input Data

The platform developer should take care to ensure that the power and performance are not heavily dependent on the input data being utilized by a particular task. If there is such dependency, the developer should take additional steps to characterize that data (e.g., based for example on the density of 1's in the data), and to adjust tuning accordingly based on the datasets.

We investigated the dependency on input data for the three benchmarks. We ran each benchmark using three data sets: one representing actual data (fax data, image data, and modem data for *g*3*fax*, *jpeg* and *v*42, respectively), and two being random data. Results are shown in Figure 7, for 40 different voltage and associativity configurations. The results show some dependence on input data, but overall trends being rather independent of the data.

4. Using a Tuning Environment

In this section, we highlight experiments of a user utilizing a tuning environment to minimize energy given a time constraint. We use the same three examples as earlier. We ran the approximate and exact algorithms on the *g*3*fax*, *jpeg*, and *v*42 examples, for two different time constraints (*Tcon*) of 0.08 and 0.3 seconds. These constraints were selected to represent one tight and one loose constraint across all three examples. Table 4 summarizes results. The number of evaluations (#) performed by our approximate algorithm averaged just over 7, and just over 9 for our exact algorithm. Thus, for this platform, the exact algorithm can usually be run without requiring excessive time.

The resulting best voltage (*V*) and associativity (*A*), and corresponding energy (*E*, in Joules) are shown for approximate and exact. The percentage difference in energy (*E*%) is the same for the two algorithms except for one case. The exact solutions were identical to the optimal solutions found during the exhaustive search experiments described in Section 3. The results show that the tuning environment can find the optimal configuration using less than 10 evaluations, much less than the 116 required for an exhaustive search.

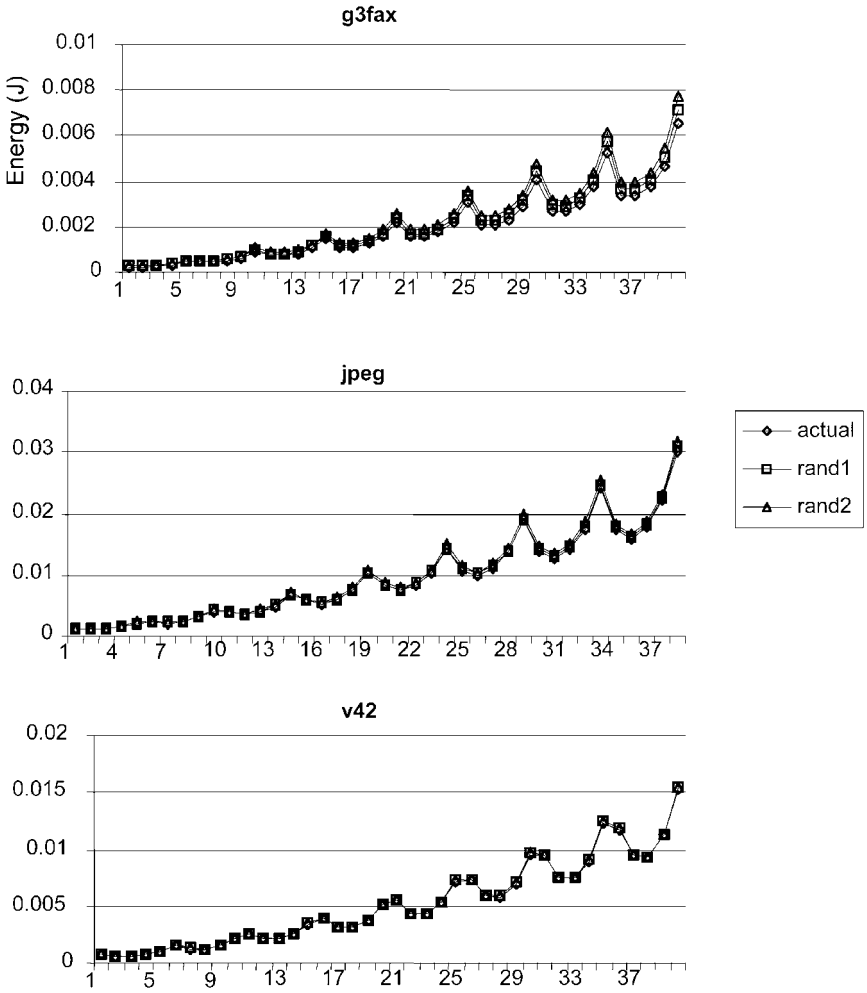

Figure 7. Energy results for different input data.

Table 4. Results of Minimizing Energy While Meeting a Performance Constraint, for Two Different Timing Constraints, Using the Approximate and Exact Algorithms

		Approximate				Exact				
Ex.	**Tcon**	**#**	**V**	**A**	**E(mJ)**	**#**	**V**	**A**	**E**	**E%**
g3fax	0.08	8	1.8	2	0.0007	10	1.8	2	0.0007	0%
	0.30	8	1.2	4	0.0004	10	1.2	4	0.0004	0%
jpeg	0.08	7	3.6	2	0.0143	9	3.6	2	0.0143	0%
	0.30	7	1.9	2	0.0040	9	1.9	2	0.0040	0%
v42	0.08	6	2.5	8	0.0044	9	2.6	4	0.0044	12%
	0.30	7	1.5	4	0.0015	9	1.5	4	0.0015	0%

Table 5. Data Showing that no Single Cache Configuration is Best for All Three Examples

Example	Tcon	A1	A2	A4	A8
g3fax	0.08	11%	0%	10%	28%
	0.30	1%	0%	10%	50%
jpeg	0.08	fail	0%	11%	28%
	0.30	25%	0%	16%	28%
v42	0.08	fail	21%	0%	10%
	0.30	55%	22%	0%	17%
Average		fail	7%	8%	27%

A platform user mapping one task to a platform need only run tuning once for that task, given the particular performance constraint and optimization criteria (e.g., energy, energy * delay). A user mapping multiple tasks to the same platform may statically allocate performance constraints to each task and then tune each task individually as above for the given optimization criteria, resulting in a static task schedule and a configuration for each task. Such runtime reconfiguration must take into account the reconfiguration time, which should be provided by the platform developer. This time will likely be dominated by the voltage scaling and clock scaling; the cache configuration takes about the time needed just to write to the configuration registers.

A platform user should be aware that different versions of the same task can yield very different performance and power results. The user may want to consider first examining different task versions before tuning a platform to the selected task. Furthermore, the user may wish to iterative tune the task to the platform and the platform to the task.

We also conducted experiments to demonstrate the usefulness of having a configurable cache in addition to configurable voltage, coupled with a tuning environment. A user without a tuning environment may not find the best configuration. In Table 5, we show the energy that would have been wasted if we had scalable voltage, but we only used a fixed cache of associativity 1, 2, 4 or 8 ($A1$, $A2$, $A4$ and $A8$, respectively), compared to an 8-way cache whose ways are shut down optimally. In other words, the 11% in the first row of column $A1$ indicates that the optimal configuration for *g3fax* with a time constraint of 0.08 seconds utilized 11% less energy than a 1-way cache. We can see by the 0% in the next column that a 2-way cache was the optimal for that example. Had we merely left all 8-ways on, the average energy wasted would have been 27%, and as much as 50% in one instance.

An observation that we can make from this data is that no one cache configuration is best for all the examples. The largest, 8-way, cache is not the most energy efficient for any of the examples. The smallest, 1-way, cache is also not energy efficient (due to numerous misses), and even fails to meet the timing constraint in two of the six cases. The 2-way cache is best for two of the examples, but the 4-way is best for the third example. For other benchmarks, the 8-way or 1-way caches might be best. Thus, we can see the importance of including configurable cache in platforms with configurable voltage, and the importance of a platform user performing the tuning step properly.

5. Conclusions

Adding configurability to platforms allows a user to tune a platform to a task's runtime profile and to power and performance constraints, making such platforms more widely-applicable and hence lowering their costs due to volume production savings. Voltage scaling and configurable caches represent two increasingly popular forms of platform configurability. We showed that combining these two features extends the meaningful configuration space considerably. We described methods to adapt a platform to particular tasks and introduced tuning algorithms for several common situations. Future work includes considering more highly-configurable caches as well as additional configurable parameters, and possibly even dynamic determination of reconfiguration values by a real-time kernel.

References

1. Albonesi, D. H. Selective Cache Ways: On-Demand Cache Resource Allocation. *Journal of Instruction Level Parallelism*, May 2000.
2. Balasubramonian, R., D. Albonesi, A. Buyuktosunoglu, S. Dwarkadas. Memory Hierarchy Reconfiguration for Energy and Performance in General-Purpose Processor Architectures. *Proceedings of Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2000.
3. Brooks, D., V. Tiwari, M. W. Martonosi. A Framework for Architectural-Level Power Analysis and Optimizations. *Proceedings of Annual International Symposium on Computer Architecture*, June 2000.
4. Burd, T. D., T. A. Pering, A. J. Stratakos, and R. W. Brodersen. A Dynamic Voltage Scaled Microprocessor System. *IEEE International Solid-State Circuits Conference*, November 2000.
5. Burger, D., and T. M. Austin. The SimpleScalar Tool Set, Version 2.0. University of Wisconsin-Madison Computer Sciences Department Technical Report #1342, June 1997.
6. Edler, J., and M. D. Hill. Dinero IV Trace-Driven Uniprocessor Cache Simulator, http://www.cs.wisc.edu/~markhill/DineroIV.
7. Geppert, L., and T. S. Perry. Transmeta's Magic Show. *IEEE Spectrum*, vol. 37, no. 5, pp. 26–33, May 2000.
8. Givargis, T. D., F. Vahid, and J. Henkel. System-Level Exploration for Pareto-Optimal Configurations in Parameterized System-on-a-Chip. *Proceedings of the International Conference on Computer-Aided Design*, Nov. 2001.
9. Halfhill, T. R. Transmeta Breaks ×86 Low Power Barrier. *Microprocessor Report*, pp. 9–18, Feb. 2000.
10. Hong, I., M. Potkonjak, and M. B. Srivastava. On-Line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor. *International Conference on Computer-Aided Design*, Nov. 1998.
11. Inoue, K., T. Ishihara, and K. Murakami. Way-Predicting Set-Associative Cache for High Performance and Low Energy Consumption. *International Symposium on Low Power Electronics and Design*, 1999.
12. Malik, A., B. Moyer, and D. Cermak. A Low Power Unified Cache Architecture Providing Power and Performance Flexibility. *International Symposium on Low Power Electronics and Design*, June 2000.
13. Nandi, A., R. Marculescu. System-Level Power/Performance Analysis for Embedded Systems Design. *Design Automation Conference*, pp. 599–604, 2001.
14. Rae, A., and S. Parameswaran. Voltage Reduction of Application-Specific Heterogeneous Multiprocessor Systems for Power Minimization. *ASP-DAC*, pp. 147–152, 2000.
15. Pering, T., T. Burd, and R. Brodersen. The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms. *International Symposium on Low Power Electronics and Design*, Aug. 1998.
16. Tiwari, V., S. Malik, and A. Wolfe. Power Analysis of Embedded Software: A First Step Toward Software Power Minimization. *IEEE Transactions on VLSI Systems*, vol. 2, no. 4, pp. 437–445, 1994.
17. Veidenbaum, A. V., T. Weiyu, R. Gupta, A. Nicolau, and J. Xiaomei. Adapting Cache Line Size to Tas Behavior. *International Conference on Supercomputing*, June 1999.