# Power-aware Partitioned Cache Architectures [*]

S. Kim, N. Vijaykrishnan, M. Kandemir, A. Sivasubramaniam, M. J. Irwin and E. Geethanjali
Dept. of Computer Science and Engineering
Pennsylvania State University, PA 16802

## ABSTRACT

This paper focuses on partitioning the cache resources architecturally for energy and energy-delay optimizations. Specifically, we investigate ways of splitting the cache into several smaller units, each of which is a cache by itself (called *subcache*). Subcache architectures not only reduce the per-access energy costs but can potentially improve the locality behavior as well. We present a unified framework for designing, implementing and evaluating different subcache architectures. Different techniques for data placement, subcache prediction, and selective probing are proposed and evaluated using a diverse set of applications. The results show that intelligent subcache mechanisms proposed in this paper are effective.

## 1. INTRODUCTION

An important trend in low power hardware design is the partitioning of hardware components into smaller and less energy consuming components [4]. The selective disabling of unused components is an effective mechanism for reducing energy consumption. Partitioning has been used for caches for both performance and energy considerations. A large cache is broken down into smaller *subbanks* to reduce the wiring and diffusion capacitances of the bitlines as well as the wiring and gate capacitances of the wordlines used to activate the memory cells. The reduced capacitance helps lower the dynamic energy consumption when accessing the caches.

Existing approaches have looked at partitioning the caches at the circuit level and enabling/disabling these subbanks at the architectural level from both the performance and energy viewpoints [1]. In contrast, this paper focuses on partitioning the cache resources architecturally for energy and energy-delay optimizations. Specifically, we examine ways of splitting the cache into several smaller units, each of which is a cache by itself (called *subcache*), and selectively activating the one holding the data upon a memory reference. The partitioning at the architectural level enables more sophisticated data placement and data probing mechanisms than those available at the circuit level. This is because at the architectural level, the application characteristics such as spatial and temporal data locality patterns can be accounted for.

This paper presents a unified framework for designing, implementing and evaluating different subcache architectures. Novel techniques for placement, prediction and probing are proposed and evaluated using a diverse set of applications from the MediaBench and SpecJVM98 benchmark suites. While there has been substantial work in architectural-level cache partitioning strategies from a performance perspective [5, 7], to our knowledge, this is the first paper that provides an extensive evaluation of different subcache architectures from an integrated energy-performance perspective. While the subcache architectures are applicable to both instruction and data caches, we primarily focus our experimental evaluations on the data cache.

The rest of this paper is organized as follows. Section 2 presents the design space for subcache organizations and explores data placement, subcache prediction and probing techniques for specific subcache architectures. Section 3 describes the experimental setup and is followed by an energy-performance evaluation of the different subcache architectures. Section 4 presents conclusions.

## 2. SUBCACHE ARCHITECTURE

### 2.1 A Taxonomy for Subcache Architectures

To explore tradeoffs between different ways of partitioning a given cache memory space into subcaches, we first define our exploration space, and then discuss the influence of each parameter in detail. We define a subcache-based system at a given level in the cache hierarchy using a tuple (N,T,PL,PR,RP,DP) where N gives the number of subcaches, and T is an array of size N that is used to define the *topology* of each subcache. More specifically, each element of T is of the form (C,l,a,ndwl,ndbl,e), where C, l, and a give, respectively, the capacity, line (block) size, and associativity of the corresponding subcache. ndwl and ndbl denote the number of wordline and bitline divisions that determine the number of subbanks. The e parameter denotes the energy-efficient features present in the subcache (e.g., way-prediction [6]). In all our experiments, the number of subbanks are determined using the timing model proposed

[*]This work was supported in part by NSF grants CCR-0073419 and CCR-0082064

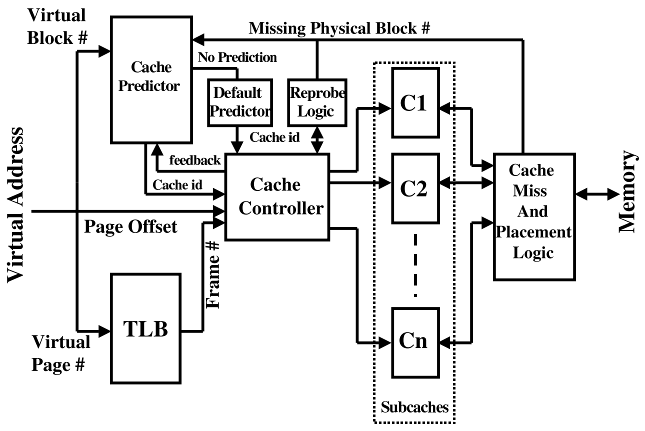

**Figure 1: Subcache architecture.**

in [9]. Note that N and T completely define the physical characteristics of the subcaches in the system. The subcache architectures that we study in this paper can be broadly divided into two categories. In *homogeneous* systems, all the subcaches have exactly the same topology (T) whereas in *heterogeneous* systems the subcaches may have different topologies.

PL is termed as the *placement policy*, and indicates how an item brought from memory is placed into the subcache system. More specifically, it is used to select the subcache into which the data is placed. Once a subcache is selected, the exact location of the data within the subcache is determined by its own topology. A wide variety of implementation choices exists for the placement policy. Selecting a good placement policy has both energy and performance implications, and its effectiveness usually depends on the amount of past history information maintained by the system.

PR is called the *cache prediction (probing) policy* and defines the strategy used to probe for the required data item in the cache system. One can come up with a wide range of probing policies. If the prediction fails, the other subcaches need to be probed (incurring performance and energy penalties). This policy is termed as the *re-probing policy* (denoted RP). Probing and re-probing schemes are closely tied to each other. Both policies could use strategies such as most-recently-used subcache (MRU), search-all-subcaches (denoted All), or history-based policies.

DP is the policy that will be activated by default for both probing and re-probing if a prediction cannot be made. Our current implementations set the default probing (DP) mechanisms to All.

## 2.2 Subcache Organization

Figure 1 shows a possible hardware strategy for implementing and evaluating numerous subcache organizations. The proposed design strives to derive an energy-efficient organization without significantly impacting performance. Normally, the virtual address (VA) generated by the CPU is fed to the TLB to find out the physical frame number, after which a cache lookup is done (we are considering a physically addressed cache). In a subcache based system, additional logic is needed to examine the address before selecting (activating) the appropriate subcache. The decision (denoted PR in the previous section) to determine which subcache needs to be activated for the first probe is done by a logic, called the *cache predictor*. If the cache predictor is introduced after the physical address translation, there is the danger of extending the critical path for data references. Instead, we propose that this operation be performed concurrently with the TLB lookup, in which case the logic will have to work with a virtual address. The output of the cache predictor will be either a *subcache id* (which will be used by the *cache controller* for the probe), or will be a *no prediction* (if it cannot make any certain determination). In the latter case, a *default predictor* (that implements DP) is used to select the cache for activation (e.g. activate-all-caches, MRU, etc.).

Based on the cache predictor output, the cache controller can activate the appropriate subcaches. The cache controller issues the read/write operation to the selected subcache, by sending it the physical frame number and page offset. This could either hit or miss in the selected subcache. If it hits, the access is complete. Otherwise, the cache miss logic informs the *re-probe logic* (that implements RP), which determines the next subcaches to probe. The next probe could either be to an individual subcache, or could be to all the remaining subcaches. The re-probe logic is active until the data is found or is determined not to be in the on-chip subcaches.

When all the re-probes fail (i.e., the main memory needs to be accessed), the *cache miss and placement logic* takes over and brings the block from main memory. In the process, one of the blocks may need to be evicted. It may be necessary to update information maintained by the cache predictor and re-probe logic units at this time. The cache predictor may also be updated whenever there is a cache hit but it has not made a prediction for it.

## 2.3 Implementation of Placement, Prediction and Re-probing Mechanisms

**Placement Strategies:** The placement policies determine the target subcache for a block brought from memory upon a miss. We investigate four different placement strategies: *Random, Least-Recently-Used (LRU), Spatial-Temporal (ST)* and *Modified-Spatial-Temporal (MST)*. The Random policy is being considered to illustrate the importance of an effective strategy. We select one of the subcaches at random for placing the block. The LRU policy places the incoming block into the subcache that was least recently used.

We investigate two different versions (ST and MST) of subcache placement mechanisms for heterogeneous subcache systems. Our ST scheme categorizes all cache lines into three groups: spatial, temporal, and bypass (non-spatio-temporal). A locality prediction based on recording the stride between consecutive data accesses is used in determining this categorization of the cache line. The data is then placed into one of the two subcaches: a spatial subcache with larger cache lines or a temporal subcache with smaller cache lines. The proposed scheme is similar to that proposed in [5]. The difference is that while the spatial and temporal data are stored in their respective subcaches, the bypass data is cached in the temporal subcache instead of bypassing the caches as is done in [5]. This modification is done as the performance is observed to be better for our target suite and configurations. Instead of fixing the location for the bypass data, it may be better to dynamically choose either the spatial or temporal subcache for better

**Table 1: Different subcache configurations used in the experiments.**

| Name | PL/PR/RP/N | Assoc. | Predictor |
|---|---|---|---|
| Direct | -/-/-/1 | direct | - |
| Random | Random/MRU/All/2 | direct | 1 |
| Split-2 | LRU/MRU/All/2 | direct | 1 |
| Split-4 | LRU/MRU/All/4 | direct | 1 |
| 2-way-predict | -/WP/All/1 | 2-way | 1 |
| 4-way-predict | -/WP/All/1 | 4-way | 1 |
| CIB | LRU/CIB/-/4 | direct | BS |
| 2-way | -/-/-/1 | 2-way | - |
| 4-way | -/-/-/1 | 4-way | - |
| ST-Direct | ST/All/-/2 | direct | LPT |
| ST-2-way | ST/All/-/2 | 2-way | LPT |
| MST-Direct | MST/MRU/All/2 | direct | LPT |
| MST-2-way | MST/MRU/All/2 | 2-way | LPT |
| MST+CIB | MST/CIB/-/2 | 2-way | LPT, BS |

performance. This is the basis of the modified ST scheme (referred to as MST), which monitors the number of spatial and temporal misses and puts the data in the subcache with a fewer number of misses (for better load balance).

**Subcache Prediction (First Probe) Strategies:** The subcache probing strategies are used to select the subcache for the first probe. We investigate three different probing strategies. The first strategy, All, accesses all subcaches concurrently. This does not require any additional prediction logic and does not provide any opportunities for energy savings during the probing stage. The second strategy, MRU/WP, accesses the most recently used subcache/way first. This MRU/WP information can be maintained in a single register that is updated with the subcache/way id on every access. The third probing strategy uses a *cache identifier buffer* (CIB) to hold a list of most recently generated virtual addresses and the corresponding physical subcaches (ids) holding those blocks. The use of virtual addresses is to allow concurrent operation with the TLB lookup rather than imposing this logic in the critical path. Whenever the CIB is not able to make a prediction (lookup fails), a default predictor (such as All) is employed. The CIB entries are updated whenever the corresponding cache line is accessed and evicted by the cache miss and placement logic.

**Subcache Re-Probing Strategies:** In order to reduce re-probe penalty on the first probe misses, all subcaches other than the one probed in the first access are accessed simultaneously in all our experiments. Configurations that use single monolithic cache and the CIB probing mechanism do not require a re-probe strategy.

## 3. EXPERIMENTAL RESULTS

The details of the evaluated configurations are provided in Table 1. 256 entries are used in the direct-mapped CIB (BS), and the direct-mapped locality prediction table (LPT) has 64 entries. MRU/WP uses just a single entry to maintain the most-recently-used subcache/way. We consider traditional single cache configurations, namely, direct-mapped (Direct), 2-way set associative (2-way), and 4-way set associative (4-way). These caches only provide circuit-level energy optimizations using sub-banking. We also consider two energy-efficient single cache configurations, namely, way prediction in a 2-way (2-way-predict) and 4-way (4-way-predict) set associative cache. In these caches, the way prediction (WP) mechanism is used to probe only one of the ways (by keeping track of the most recently used way), thus saving energy in the unaccessed ways beyond the circuit-level optimizations. The difference between our subcache prediction and traditional way-prediction schemes is that the prediction granularity in the former scheme is a *subcache* whereas that of the latter is a *way*. We compare these single cache configurations with nine different subcache organizations. These include five heterogeneous configurations (ST-Direct, ST-2-way, MST-Direct, MST-2-way and MST+CIB) and four homogeneous configurations (Random, Split-2, Split-4 and CIB). In both the homogeneous and heterogeneous configurations, the subcaches are of equal size. Except for the spatial cache (in ST and MST schemes) which has a line size of 64 bytes, all other caches and subcaches used a line size of 32 bytes. For a fair comparison, in all the configurations, the total subcache capacity including the overheads is equal to the size of a 32K direct-mapped cache.

We feed the data references from the execution of applications using the *Shade* binary instrumentation tool [2] to the cache simulator. Our cache simulations use the *cachesim5* simulator available in the *Shade* suite for the direct-mapped and set associative configurations. All other schemes are built on top of the *cachesim5* code. We use the applications from two benchmark suites, SpecJVM98 and the MediaBench. The energy consumption of the subcache configurations is evaluated using 0.18 micron technology parameters and a 2V supply voltage with the cache energy models proposed by Kamble et. al. [8]. Off-chip energy consumption is based on measured values reported for a 8MB RDRAM module [10]. The performance evaluation is done based on the number of cycles spent in servicing a memory request. On-chip accesses are modeled to perform single-cycle accesses and off-chip accesses have a latency of 100 cycles.

**Results:** We give experimental data on performance, energy, and energy-delay product. For purposes of clarity and space limitations, rather than presenting the results of each application, we present results averaged over all the applications for each benchmark suite. The results are summarized in three normalized graphs for each benchmark type (one for energy, one for performance, and one for energy-delay product). In obtaining these graphs, first, the values for each configuration are normalized with respect to the corresponding direct-mapped cache configuration, and then averaged over all benchmarks in the suite in question.

**Performance:** There are two factors affecting the overall performance of a given configuration. First is the capability of reducing the number of cache misses, and second is the effectiveness of the probing strategy that influences re-probing penalty. As can be observed from Figure 2, the MST+CIB provides an effective subcache management scheme as it supports both efficient placement and probing strategies. The use of MST is observed to improve the overall performance due to the first factor as it allows better cache utilization by exploiting the inherent data reuse, and by placing the data into appropriate subcache depending on the type of locality it exhibits. Overall, we observe that CIB is an effective probing strategy, making the subcache schemes that employ CIB for *probing* (i.e., CIB and MST+CIB) very competitive. When the program exhibits good locality, CIB has a high probability of making a prediction.

**Energy:** Two factors influence the energy consumption of a given subcache architecture. First, improved performance can lead to reduction in number of off-chip accesses that are more expensive from an energy viewpoint as compared to on-chip accesses. Second, the effectiveness of the

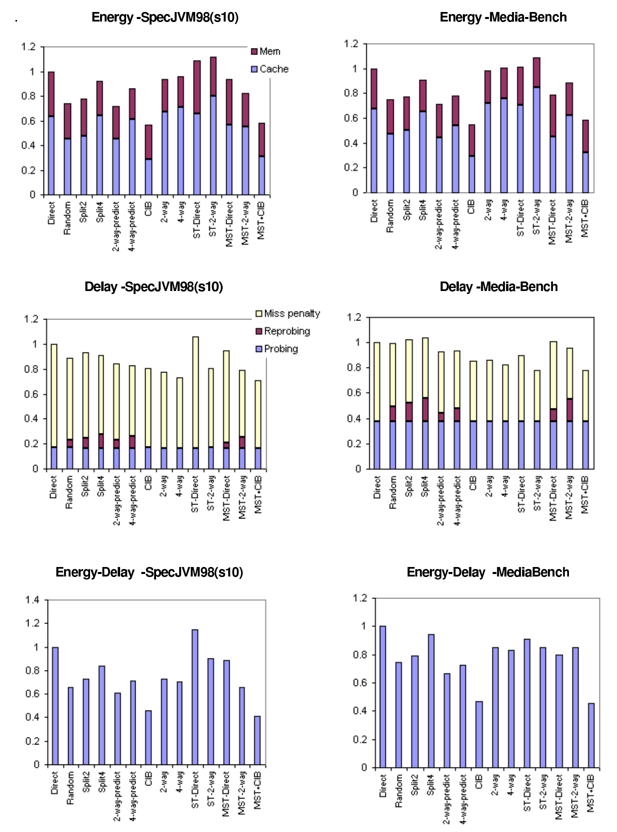

**Figure 2: Performance, energy, and energy-delay results of MediaBench and SpecJVM98 (s10 datasets) suites for different subcache configurations.**

probing strategy determines the number of subcaches accessed per data reference which has a direct consequence on energy. The energy consumption is profiled for the subcache system (including the predictors), bus, and off-chip memory components. We first observe that the cache energy is a dominant part of the overall memory system energy across all the subcache and traditional cache organizations. Since the CIB scheme employs four subcaches, its per access energy cost is better. It is also able to predict reasonably well as observed from the performance results. The combination of these two factors and the dominance of the cache energy in the overall picture make this the least energy consuming alternative. Split-4 has the same per access energy, but since it predicts poorly, its energy consumption is much higher.

The use of ST subcache configuration can have two effects on the energy consumption. First, it can reduce the off-chip energy consumption by improving the data locality as in the case of MediaBench (See Figure 2). Second, since it employs two subcaches accessed in parallel rather than a single cache, the energy consumption per access increases. It must be noted that the reduction in cache energy with size is sublinear. Since the reduction in off-chip energy is relatively smaller compared to the higher per access energy cost, it makes ST the most energy consuming version. The CIB-based schemes (CIB and MST+CIB) give much more energy savings than 2-way-predict and 4-way-predict schemes. The energy trends between MediaBench and SpecJVM98 are quite similar, and the CIB-based schemes consistently give around 40% energy savings compared to a unified direct-mapped cache.

**Energy-Delay:** It is also important to quantify the trade-offs between the energy and performance optimizations of the proposed schemes. In many mobile systems both of these issues are of critical importance, making energy-delay product an important parameter to evaluate different design alternatives. We observe that due to their success in selectively activating the right subcache, the CIB-based schemes give the best energy-delay product. This observation is valid across the different applications evaluated as observed from Figure 2. Among the subcache configurations, Split-4 and ST-direct perform the worst and also worse than traditional set associative caches. The circuit-based energy-efficient schemes (2-way-predict and 4-way-predict) have an energy-delay product that falls between the CIB-based schemes and the other subcache configurations studied.

## 4. CONCLUDING REMARKS

This paper has shown that with intelligent subcache architectures (especially the MST+CIB scheme proposed in this paper), we can get 23% and 28% improvement in memory system performance, 42% and 42% improvement in memory system energy, and 55% and 58% improvement in energy-delay product, as compared to a single 32K direct-mapped cache for the MediaBench and SpecJVM98 applications respectively.

## 5. REFERENCES

[1] D. H. Albonesi, Selective cache ways: On-demand cache resource allocation, In Proc. *The 32nd International Symposium on Microarchitecture,* November 1999.

[2] R. F. Cmelik and D. Keppel, Shade: A fast instruction-set simulator for execution profiling, Tech. Rep. SMLI TR-93-12, Sun Microsystems Inc, 1993.

[3] A. Chandrakasan, W. J. Bowhill and F. Fox, *Design of High-performance Microprocessor Circuits,* IEEE Press, 2001

[4] J. L. Cruz, A. Gonzalez and M. Valero, Multiple-banked Register File Architecture, In Proc. *International Symposium on Computer Architecture*, June 2000.

[5] A. Gonzales, C. Aliagas and M. Valero, A data cache with multiple caching strategies tuned for different types of locality, In Proc. *International Conference on Supercomputing,* July, 1995.

[6] K. Inoue, T. Ishihara and K. Murakami, Way-predicting set-associative cache for high performance and low energy consumption, In Proc. *International Symposium on Low Power Electronics and Design*, 1999.

[7] T. L. Johnson and W. W. Hwu, Run-time adaptive cache hierarchy management via reference analysis, In Proc. *Annual International Symposium on Computer Architecture*, 1997.

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

[9] S. Wilton and N. Jouppi, An enhanced access and cycle time model for on-chip caches, Technical Report 93/5, DEC WRL Research report, 1994.

[10] 128/144-MBit Direct RDRAM Data Sheet, Rambus Inc., May 1999.