An Energy Efficient Parallel Architecture Using Near Threshold Operation

Ronald G. Dreslinski, Bo Zhai, Trevor Mudge, David Blaauw, and Dennis Sylvester
University of Michigan - Ann Arbor, MI
{rdreslin, bzhai, tnm, blaauw, dennis}@eecs.umich.edu

Abstract

Subthreshold circuit design, while energy efficient, has the drawback of performance degradation. To retain the excellent energy efficiency while reducing performance loss, we propose to investigate near subthreshold techniques on chip multiprocessors (CMP). We show that logic and memory cells have different optimal supply and threshold voltages, therefore we propose to allow the cores and memory to operate in different voltage regions. With the memory operating at a different voltage, we then explore the design space in which several slower cores clustered together share a faster L1 cache. We show that an architecture such as this is optimal for energy efficiency. In particular, SPLASH2 benchmarks show a 53% energy reduction over the conventional CMP approach (70% energy reduction over a single core machine). In addition we explore the design trade-offs that occur if we have a separate instruction and data cache. We show that some applications prefer the data cache to be clustered while the instruction cache is kept private to the core allowing further energy savings of a 77% reduction over a single core machine.

1. Introduction

Due to its high energy efficiency subthreshold design has been recently proposed for use in low performance applications. For example, Zhai et al. [1] proposed a low-end sensor network processor and Wang and Chandrakasan [3] proposed subthreshold use for an FFT engine. However, the drawback of subthreshold design is that the increased energy efficiency comes at the cost of performance loss. Thus previous papers have only targeted low end applications. Our intention is to use the energy efficiency of subthreshold or near threshold designs to target parallelizable embedded applications requiring higher performance, but where battery life is important.

Previous work by Zhai et al. [4] and Calhoun [5] has shown that for a CMOS digital circuit there exists an energy optimal supply voltage (V_{min}) below which energy consumption increases because of exponentially increased propagation delay and leakage energy. V_{min} usually occurs in the subthreshold region. However, these papers did not address how to choose the threshold voltage (V_{th}) to further improve energy savings. Zhai et al. [2] showed that by properly controlling V_{th} to reach the energy optimal voltage a greater savings can be achieved. We extend that work and investigate, in detail, architectural choices that can be used to exploit this property.

Zhai et al. [1] shows that, due to the different activity factors and leakage rates for memory cells and logic, the V_{min} of the processor and memory are usually different. As a result, operating the entire system under a single V_{dd} leads to sub-optimal energy efficiency. Therefore we propose using a chip design that has two separate V_{dd}/V_{th} domains for the core and memory.

However only changing the V_{th} does not solve the issue of performance loss from aggressive voltage scaling. So we propose to employ multiple cores in the near threshold regime, where we get the best performance for energy trade-off. We explore separate control of V_{dd} and V_{th} for the core and the memory, where memory is allowed to operate at speeds both slower and faster than the core it is attached to. In particular, we create a cluster that has several slower cores connected to the same faster cache.

Clusters have advantages compared to the conventional CMP approach. Applications that have high communication to computation ratios can share data with other cores in the cluster without the coherence overhead of communicating over the bus that connects the different L1's. However, the cores are now contending for the same cache space, which may result in effectively smaller cache sizes due to conflict and capacity misses generated by the other cores within the cluster. L1 sharing also requires a bus between the cores and the L1. Having more cores within a cluster increases the size and capacitance of this bus. We investigated all the major factors that could affect the system energy efficiency, such as L1 cache size, the cluster size, total number of clusters and the selection of V_{dd} and V_{th} within a cluster. Architectural simulation together with circuit-level modeling shows that for most of the SPLASH2 [6] benchmarks the energy optimal point is 2 cores in a cluster. This configuration provides about a 53% increase in energy efficiency over the conventional CMP design.

In addition we explore the possibility of separately clustering the instruction and data caches and explore the impact this has on different applications. We find that due to the low miss rate and high access rate of the instruction cache, further energy savings can be found by keeping small private instruction caches for each core and a clustered data cache. Using this technique we can increase energy efficiency further to about a 59% reduction over the conventional CMP design.

This work is a detailed extension of the architectural analysis of earlier work which focused on proper V_{th} and V_{dd} selection for clustered MP architectures[2]. This work furthers that work by providing a deeper analysis of the architectural trade-offs and furthers the research by exploring split I/D caches.

The paper is organized as follows: Section 2 explains the advantage of the near threshold design and the impact of threshold voltage selection on energy efficiency. We then introduce the proposed architecture in Section 3. Section 4 lays out our benchmark selection. Then Section 5 details the energy modeling of a complete system and Section 6 details the architectural simulation results of the SPLASH2 benchmarks. In Section 7 we present the impact of splitting the L1 cache into instruction and data caches and hit under miss policies. Finally, Section 8 and Section 9 present related work and concluding remarks. All technology numbers are from an industrial 0.13um CMOS technology.

2. Near Threshold Operation

Zhai et al. [4] identified the supply voltage at which the minimum energy is achieved, V_{min}. They also showed that minimum energy consumption at that voltage, E_{min}, is independent of the threshold voltage, $V_{th,}$ if V_{min} is below V_{th}. In other words, we can increase the speed of a subthreshold circuit by reducing V_{th} while maintaining the same energy consumption. However, V_{min} varies with the activity rate of the circuit. Less active components such as an SRAM/cache usually have a higher V_{min} than a core.

Zhai et al. [2] looked in detail at how V_{th} and V_{dd} impact V_{min}. There are three key points from that work that motivate the architectural study here. These points can be identified from figure Figure 1 where we present the V_{min} for different choices of V_{dd} and V_{th}. First it is important to note that the core and memory prefer to operate at different supply voltages for their optimal operating point. This motivates the choice of a system in which we separately control the voltage for the memory system and the core. The second key result follows from this observation; since the operating voltage of the memory is higher and the activity factor lower, the memory prefers to operate at speeds faster than the core. This allows us to investigate more interesting architectures where several slower cores are connected to a single faster cache forming what we call a cluster. The third result to be drawn from Figure 1 is that by changing Vth we can achieve additional energy gains when operating in the near threshold region. For further in depth analysis on this please refer to the work done in [2].

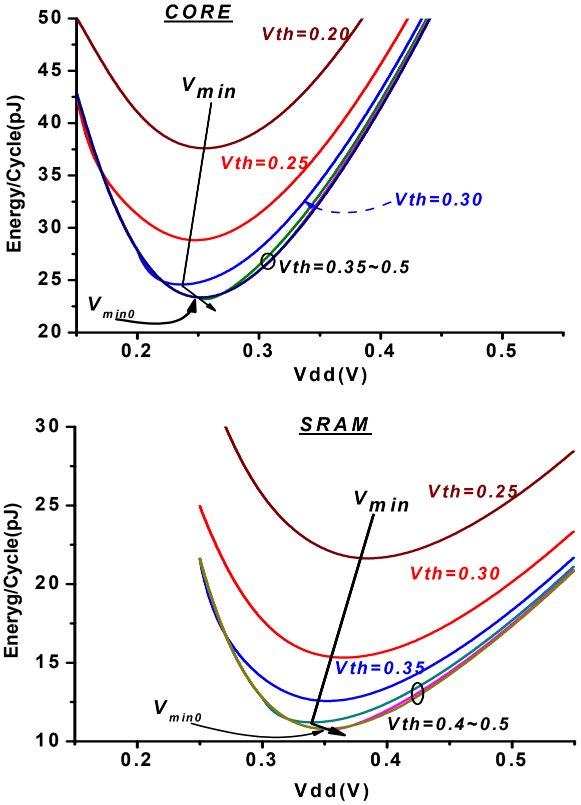


Figure 1. Energy-V_{dd} for the core and the SRAM with different V_{th}s

3. Proposed Near Threshold Architecture

3.1. New Memory Architecture

Traditional computer design is usually limited by the SRAM/cache speed leading to conventional designs in which the caches run at the same speed or slower speeds than the core. However, the picture changes in the near threshold domain because we can now take advantage of the V_{dd} and V_{th} knobs, as discussed in Section 2, and have the memory module operate at a speed faster than the core. In the subthreshold regime, a 100mV boost in V_{dd} from 300mV to 400mV can bring almost 10X performance speedup according to silicon measurements in [1]. We can also tune the V_{th} similarly to adjust speed. However, this is not feasible in the superthreshold regime because driving current depends on V_{dd}/V_{th} following the α-power law [7].

We further investigated the use of multiprocessing techniques in conjunction with voltage scaling techniques to reduce energy consumption. The conventional way that multiprocessing is implemented is illustrated in Figure 2 (a) where there is one cache per core. In this paper we propose a new micro-architecture shown in Figure 2 (b) where there are several cores sharing one local cache forming a “cluster”. The local cache serves k cores by running k times faster than each individual core. We achieve this by assigning proper V_{dd} and V_{th} to the cores and the SRAM/caches. To simplify the

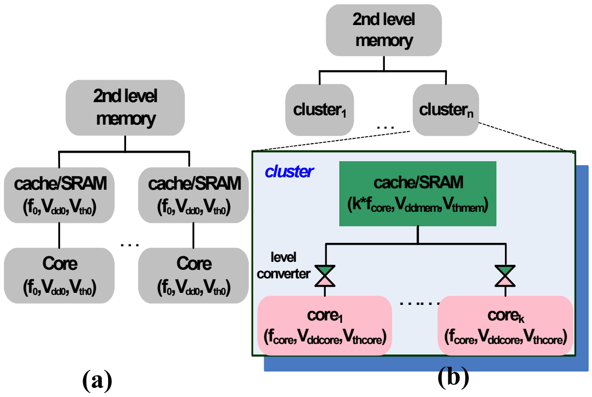

2nd level memory
cache/SRAM (f_0,V_{dd0},V_{th0})
cache/SRAM (f_0,V_{dd0},V_{th0})
Core (f_0,V_{dd0},V_{th0})
Core (f_0,V_{dd0},V_{th0})
2nd level memory
cluster$_1$
cluster$_n$
cluster
cache/SRAM ($k*f_{core}$,V_{ddmem},V_{thmem})
level converter
core$_1$ (f_{core},V_{ddcore},V_{thcore})
core$_k$ (f_{core},V_{ddcore},V_{thcore})
(a)
(b)

Figure 2. Conventional (a) and proposed (b) MP architectures

problem, we assume that all the cores are running at the same speed and same V_{dd} and V_{th}. But the V_{dd}/V_{th} between the cores and the SRAM/cache could be different. The clusters are connected to the same next level memory, which could be an on-chip cache or an off-chip DRAM.

Because the cores and the memories could potentially operate at different supply voltages, level converters will be needed in between the cores and memories. Subthreshold level converters have been shown feasible in [1]. Considering the V_{dd} space we are exploring, the delay of the level converter is minimal compared to the critical delay of the core and the memory. In addition, since we are now connecting multiple cores to a single L1, the tightly coupled nature of the L1 will be removed and a bus will be needed to connect the cores and the cache. This bus will become larger and expend more energy as the number of cores within the cluster is increased.

3.2. Architectural Trade-offs

This proposed architecture takes advantage of several different trade-offs within the design. The next few sub sections break down each of the architectural parameters that we are able to vary and describes the trade-offs being made. Each of the different trade-offs yield an optimal energy point for a given benchmark. In Section 6 we will do a complete analysis of the parameters and their interactions for several different benchmarks to find the energy optimal point for each.

3.2.1. L1 Cache Size

The size of the L1 is an important factor in controlling the overall system energy. Traditionally we prefer a larger L1 cache because it results in more hits and therefore a shorter average memory access time. When considering energy consumption, this shorter average access time means that the CPU itself can complete the work in fewer total cycles. Since it takes fewer total cycles we can slow down the system and still meet the timing deadline with less energy expended. In addition the higher hit rate also means fewer accesses to the L2. Since the L2 is running at nominal voltage it has a high energy cost per access. Having fewer accesses results in less energy and greater opportunity for the L2 to take advantage of techniques like drowsy caches [16].

On the other hand increasing the size of the L1 also increases the energy per access. This happens because the tag lookup requires a larger structure, and the busses that run through the cache become larger with more capacitance. We use a muxed based memory to reduce the increase in access energy by banking memory to reduce the size of the internal buses. This adds complexity to the cache because muxes are inserted, but it reduces the energy increase compared to a single large cache bank. Even though we use techniques to limit the increase in energy, there is still additional energy consumed per access. This additional energy per access will result in more energy being consumed for larger caches.

The trade-off here is really to balance the L1 access energy with both the decrease in L2 energy and the faster completion time. This parameter is highly dependent on the access pattern and data set size of the application. Applications with small working sets tend to prefer smaller caches because there tends to be fewer L1 misses with smaller working sets.

3.2.2. Cluster Size

The size of the cluster is also an important factor, and highly depends on the benchmark. The biggest benefit of increasing the cluster size can occur in applications with high communication to computation ratios. Clustering the CPUs together allows cores within a cluster to communicate through shared memory without having to traverse the bus connecting the L1's. As the number of cores in a cluster is increased a given core can reach more cores in a single cycle. The other benefit is similar if the cores are sharing data. The sharing pattern doesn't matter as much as the amount of shared data. Similar to the communication latency being shorter, data can be shared between cores within a cluster without the overhead of the coherence protocol. This way if a block was being ping-ponged between two cores, by putting them in the same cluster we lower the average access latency. This decrease in both the average communication latency and shared memory latency results in lower power in two main ways. First the average number of cycles required to complete the program is lower, and thus the cpu and system can be clocked at a slower rate and lower voltage and still meet the required execution time. This will result in lower overall energy consumption. Second the number of Snooping access that occur to other L1's is decreased. This means fewer tag lookups in all the L1 caches as well as the snoop request on the bus between the L1's.

On the other hand, as the number of cores in a cluster is increased the cores begin to contend for the same cache resources. This will result in more conflict and capacity misses unless the size and/or associativity of the cache size is increased. The trade-offs of an increased cache can be found in Section 3.2.1. Given a fixed cache size, the increased number cores will generate more L2 accesses, which are costly in terms of energy because the L2 is operating at a higher voltage. In addition the cache must be clocked at a higher rate to satisfy the requests. This higher frequency will result in the cache needing a higher voltage, thus increasing the energy required per access. Also, as we increase the number of cores within the cluster we must also connect the cores physically with a bus. Traditionally the L1 cache can be tightly coupled with the core creating a fast interface with low capacitance. With a clustered architecture a bus will need to be added to connect the cores. This bus will slow memory accesses, and the capacitance will increase with the number of cores. Driving that larger capacitance will result in increased energy consumption.

The trade-off here is the speeding up that occurs due to the sharing between cores at the cost of increasing cache contention, access energy, and bus size. This is highly dependent on the application. For example an application that has a high communication to computation ratio and a large amount of shared data that is frequently accessed by many cores will prefer a large cluster size. While independent workloads (several independent threads) that have little to no communication or shared data will prefer smaller cluster sizes.

3.2.3. Number of Clusters

The number of clusters in a system can also be classified as the number of cores in the system when we hold the cluster size constant. The benefit of having more clusters is that we can expose more parallelism. This increased parallelism means that we can divide the task among more cores and clock each of the cores at a slower frequency and lower voltage. The lower voltage and frequency will lead to energy savings.

On the other hand with increased parallelism we introduce more communication overhead between the cores and additional code to parallelize the application. This overhead results in slightly larger instruction counts and more communication across the snooping bus. In addition Amdahl's law applies to the amount of parallelism we will be able to extract in comparison to the inherently serial portion of the code. This of course means that the number of clusters is also dependent on the application that is being run in terms of the amount of parallelism that can be extracted and the amount of code overhead and communication that occurs when we parallelize the application.

4. Benchmarks

For our analysis we have chosen to use the SPLASH2 benchmark suite[6]. These benchmarks represent several different forms of parallel applications. Although we are targeting less computationally intensive workloads, i.e. MPEG decoding, we feel that the SPLASH2 benchmarks offer a wide variety of parallel execution models. The important factors to consider for any application are the amount of parallelism that is present in the application, the data set sizes, the communication to computation ratios, and the amount of data sharing that occurs. These factors play the most important role in the performance of the machine. We used the smaller SPLASH2 input sets for faster simulation time and to represent the smaller nature of the problems that we are targeting. The applications we chose to run were Cholesky (cho), FFT, FMM, LU non-contiguous blocks (lun), LU-contiguous blocks (luc), Radix (rad), and Raytrace (ray).

Although these applications are not exactly what we are targeting, we can use them to determine if clustering is beneficial and determine the performance range at which we get the best energy efficiency.

5. Simulation

Our simulation was done with the M5 simulator[15]. It was modified to allow clustering support of the L1 caches. To provide the ability for multiple cores to access the cache in the same cycle, the cache is operated at a higher frequency and the cores within the cluster are run in different phases of the cache clock. For example if there were 4 cores per cluster the cache would be clocked 4x the speed of the cores. The first core would have a rising edge of it's clock on the 1st, 5th, 9th,... clock edge of the cache. The second core would have a rising edge on the 2nd, 6th, 10th,... clock edge of the cache and so forth. This means that each core will see a single cpu cycle latency to the L1 without the need for an arbiter on the bus, or multiple ports to the cache.

5.1. Power Models

In order to properly attain energy numbers from our architectural simulation, we needed to determine a power model for all the components in the system. The following subsections will describe how we arrived at the model for each component in the system.

5.1.1. Core and L1

The core energy estimations are based upon the processor core frequency and power consumption numbers from ARM946 in [8]. Then we used the same fitted model as in Section 2 to capture the V_{dd} and V_{th} dependency of delay and leakage. The processor without caches consumes 86mW while running at 233MHz with a nominal V_{dd} of 1.2V. The cache power/energy numbers of different sizes were extrapolated from a memory compiler in 0.13um technology. The baseline machine is assumed to have the parameters in Table 1. 64kB of unified data and instruction L1 cache is chosen as a reasonable number considering the problem size of the SPLASH2 benchmark applications [6].

Previous work [9] has shown that current sensitivity to subthreshold variation increases dramatically with reduced V_{dd}. Previous work [3][10][11][12][13] has illustrated successful SRAM designs that operate in subthreshold regime. However, all these works result in an area overhead. Part of the reason for the area overhead is that larger channel area helps suppress random dopant fluctuations (RDF), the dominating factor in the subthreshold regime [14].

In order to factor in the design area overhead of voltage scalability for the SRAM, we carried out Monte Carlo simulations and determined the amount of up-sizing needed for the memory cells under a certain yield constraint. A exponential function is then fitted to the results and used in the rest of the paper. This up-sizing increases the physical size of the cache, which, in turn, increases both the access energy and the bus length to connect the cache. These additional increases were modeled in our evaluations.

TABLE 1. Baseline Architecture

Parameter	Value
CPU	233MHz, in-order functional model
Unified D/I L1 cache	64kB, 2-way, block size=64B
L2	2MB 8-way, latency=10

5.1.2. DRAM and L2 Design

Off-chip access to DRAM is power-hungry because of the high capacitance in the chip package and off-chip wires. Therefore, a energy-oriented design needs to have a big enough L2 cache so that it can shield off the majority of the conflicting L1 misses from accessing the off-chip memory. For our analysis we have chosen a 2MB on-chip L2 cache. Considering the size of this L2, we decide to fix its operating voltage at nominal (1.2V) instead of voltage scaling to avoid over design. As aforementioned, designing a large L2 for voltage scalability with high yield implies significant area overhead and therefore results in high switching and leakage energy.

From simulation we found that the L2 is not heavily accessed and has a much lower activity rate compared to the L1. Therefore we will design the L2 cache by using low standby leakage techniques such as drowsy cache [16]. The standby leakage is assumed to be 1/20 of that during active mode. The same argument does not hold for the L1 because the L1 is accessed considerably more frequently than the L2. The L1 is also smaller in size than the L2 cache and we are able to trade off some area for energy efficiency and voltage scalability.

5.1.3. Bus Modeling

The physical parameters are drawn from a 0.13um technology doing detailed analysis of the repeaters and sizes necessary for proper operation. Using this model we can include bus energy for both the CPU-L1 bus and the L1-L2 bus, which consists of data, address, and command fields. The length of the CPU-L1 buses scales linearly with cluster size and the length of the L1-L2 buses linearly with cluster number. Basic footprint size is taken from a commercial processor core [8]. No breakdown of bus energy is presented in the graphs, but all graphs and data referring to total energy include the bus energy numbers.

5.2. Baseline Machine

Our baseline machine is summarized in Table 1. It is a simple in-order CPU running at 233 MHz with a 64kB unified cache. For all configurations we hold the runtime to be the same as that of the baseline machine. In Section 6.6 we will look at the optimal choices at different baseline performance numbers (CPU Frequencies).

5.3. Simulation Configurations

We simulate the system by varying the cache size from 4-128kB, the number of clusters from 1-16, and the number of cores per cluster from 1-8 for each SPLASH2 benchmark. We then do a power analysis with different voltage scaling techniques. One in which we do the same V_{dd} scaling on both the L1 and core, another where we scale V_{dd} separately for the L1 and core, and the third where we scale V_{th} separately as well.

6. Results

In order to analyze the impact of the proposed architectures in energy efficient processor design, we must quantify the system performance energy trade-off using architectural simulations. We performed the simulation using the M5 Simulator [15] to determine the energy consumption for different configurations. During the cluster mode, we assume that the L1's are k times faster than the core, where k is the number of cores per cluster.

The relative latency of the components need to be scaled so as to capture the voltage scaling effect. For instance, when we have a multicore system, each core and L1 inside the system can be accordingly slowed down compared to the single-core baseline machine, since our constraint is to keep the runtime constant. In architectural simulation, this is equivalent to having a faster L2 and DRAM.

6.1. Optimal L1 Size

It is known that the size of the L1 affects the system performance in terms of average access latency, but more importantly we found that it also affects the energy efficiency of the system. In order to understand how the choice of L1 size impacted the energy performance of a system, we first performed an analysis on a supply voltage scaled uniprocessor system. With the number of parameters we intended to vary for the complete study, this was a simple way to do a first pass analysis and present some interesting results while only varying one parameter. We present the results while allowing the die size to increase as we increase the L1. A similar study was done where the die size was held constant and the L2 was made smaller to compensate the increase in the L1 and the results were similar.

With a single processor and single L1 system, we varied L1 sizes and optimized the energy consumption for each size by tuning the V_{dd} and V_{th} of each component. Figure 3 shows the energy consumption for Cholesky. We found similar trends for all the SPLASH2 benchmarks, although the optimal cache size varied across the applications. The energy consumption for the processor core, L1, and L2 are also shown. As L1 size

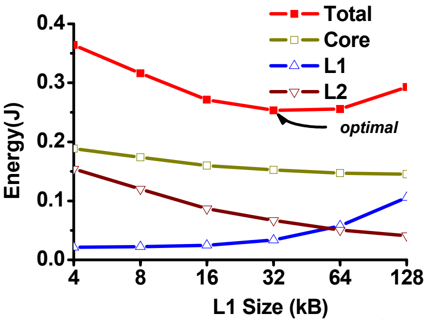
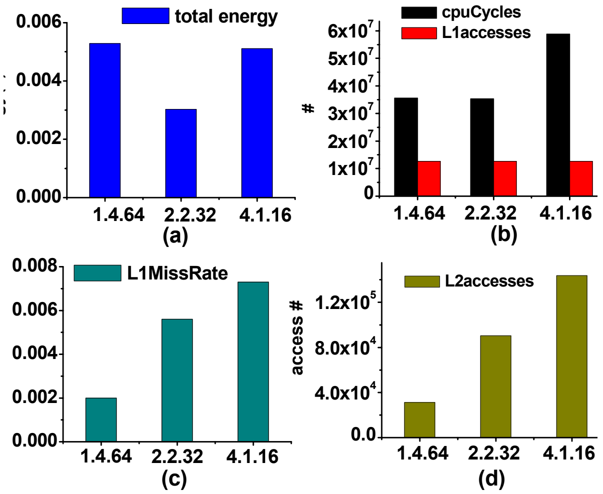


Figure 3. Optimal energy consumption for *Cholesk* when using one core and one L1



Notation: (cluster #).(cores per cluster).(L1 size per cluster in KB)

Figure 4. Three configurations comparison

increases from 4kB to 128kB, the number of accesses to the L1 remains constant because the same instruction stream and data pattern come from the processor core. However, L1 energy consumption increases with larger L1 sizes due to larger array size and larger capacitance. The other implication of various L1 sizes is the L1 performance. The L1 miss rate reduces significantly with larger L1s, which also results in a lower number of access to the L2 due to fewer L1 misses and writebacks. Therefore the L2 energy reduces with larger L1 sizes. The core's energy consumption also reduces with larger L1 sizes because of the reduced number of clock cycles that results from a higher L1 hit rate (lower average access latency).

6.2. Optimal Cluster Size

After having an understanding of how the L1 size impacted energy performance we move on to studying the impact of multiprocessing and clustering on the energy consumption. The setup for this study is described in Section 5.3. The analysis was done on the 240 configurations for each benchmark.

In order to fully understand the benefit of running in cluster mode Figure 2(b) vs. conventional connection Figure 2(a) we specifically compare three cases in which the overall die size is held constant and present the results in Figure 4. Figure 4(a) shows the energy consumption of the three candidates for Cholesky in the SPLASH2 benchmark suite. The best configuration for energy efficiency is 2 cores per cluster, with 2 clusters.

To help illustrate the reason 2 cores per cluster was optimal we break down the results in more detail in Figure 4(b-d). At 4 clusters with 1 core per cluster, a conventional CMP, the number of CPU cycles is larger than both of the clustering cases. This occurs because the average memory access latency is longer because the L1 miss rate is high due to the fact that shared memory accesses are forced to miss in the local L1 cache and snoop neighboring L1's to get the data. Now as the number of cores in a cluster is increased to 2 some of the shared memory of other cores is visible within the cluster's L1 to both cores without having to access another L1. This significantly reduces the average memory access time and results in a reduced number of CPU cycles. Meanwhile the larger shared cache within the cluster results in a higher hit rate and reduced accesses to the L2. Reducing the number of access to the L2 reduces the energy consumed by the L2 as was shown in Section 6.1.

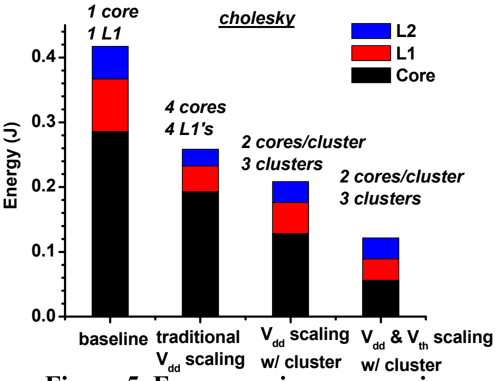


Figure 5. Energy savings comparison using various scaling methods

As we scale to 4 cores per cluster we now have reduced the L1 miss rate even further, but the energy per access of the L1 and the energy to operate the large bus to connect the cores within the cluster begin to outweigh the gains we see in reduced L2 traffic. Also, you can see that there is not a large reduction in CPU cycles. This happens because the first set of clustering encapsulated most of the memory sharing and synchronization that took place in this application.

For this study it is important to note that we chose three points at which the die size was nearly the same. Remember that there are some interconnect and memory scaling issues, but the sizes are still close. In the later studies we will present the global optimal configuration without regard to total cache or die size. We wanted to show in this study that given a fixed die size, clustering is the optimal choice for this benchmark

6.3. Various Energy Saving Modes

Now that we have shown for a fixed die size that clustering can outperform conventional CMP designs, we explore the entire design space and analyze the impact of using the V_{th} to further improve performance. We again use the same 240 configurations per benchmark as in Section 6.2. The results presented in this section are for the Cholesky benchmark, results for other benchmarks are summarized in Section 6.5. In this study we evaluate the impact of different scaling approaches. Specifically, we assume the baseline single-core single-cache machine does not do voltage scaling and runs constantly at 1.2V. The three different scaling approaches are: 1) traditional V_{dd} scaling using MP while maintaining one core per L1 cache (Figure 2 (a)), 2) V_{dd} scaling using MP but with cluster mode configuration and 3) V_{dd} and V_{th} scaling with cluster mode configuration. The runtime of all three systems is set to match that of the baseline machine.

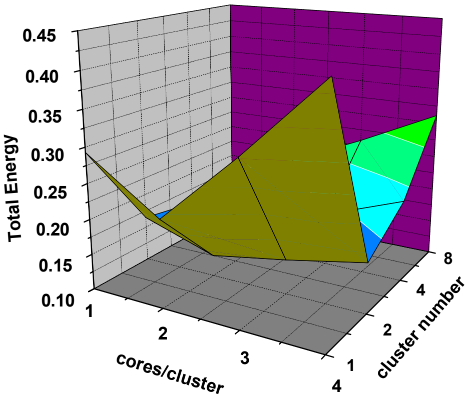


Figure 6. Total Energy for *Cholesky*

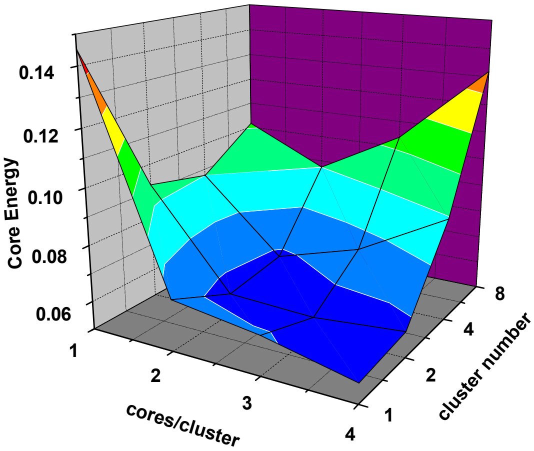


Figure 7. Core Energy for *Cholesky*

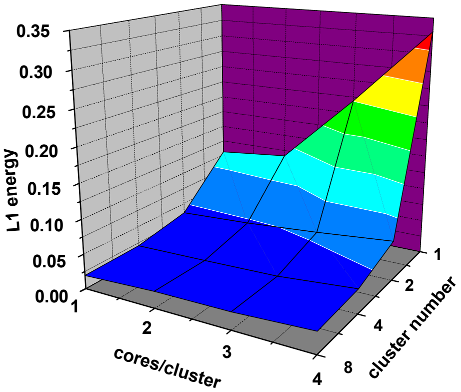


Figure 8. L1 Energy for *Cholesky*

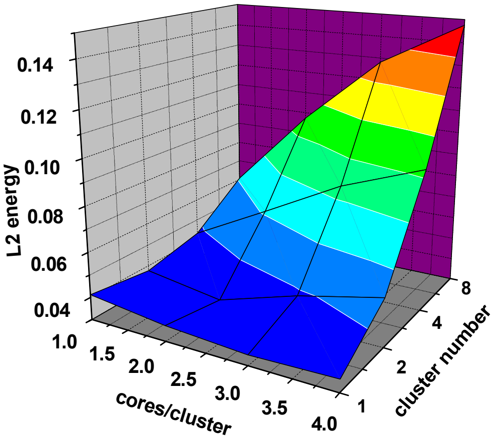


Figure 9. L2 Energy for *Cholesky*

The results are presented in Figure 5. Conventional CMP V_{dd} scaling brings us 38.1% savings over the baseline machine, but clustering cores with V_{dd} scaling results in a 18.9% improvement over traditional scaling techniques and 49.9% improvement over the baseline. Finally we show that clustered cores with both scaled V_{th} as well as V_{dd} yields a global optimal energy, and about a 53% percent improvement over using traditional scaling techniques on a conventional CMP.

The V_{dd} scaling technique with conventional CMP finds an optimum with 4 cores and 4 L1 caches at a V_{dd} of 0.8V. With clustering, the system finds an optimal with 2 cores per cluster at 3 clusters. The optimal L1 size for all 3 scaling approaches is 64kB for this benchmark. The energy optimal point was chosen without regard to die size. If given a die size constraint the energy optimal point can be found within the configurations that meet that die constraint. Section 6.2 provides an example of fixed die size analysis where three configurations with the same die size are compared.

6.4. Multi-Dimensional Analysis

To understand the entire design space further we present Figures 8, 9, 10, and 11. These figures breakdown the entire design space for a fixed total L1 size, 128kB, for the Cholesky benchmark using both V_{dd} and V_{th} scaling. In Figure 6 we present the total energy of the system, you can see that there is a well along the 2 cluster axis. This minimum represents the fact that the optimal number of clusters is 2. Additionally there is a dip along the 2 cores/cluster line, resulting in a optimal location of 2 clusters with 2 cores per cluster. To understand this better we break down the core energy in Figure 7, the L1 energy in Figure 8, and the L2 energy in Figure 9. Note the cluster number axis is in the opposite direction for Figure 8.

The core energy looks like a surface that decreases from left to right and from back to front. As you go from front to back or from left to right there is an increasing total number of cores. As we increase the number of cores, we can run each of them at a lower voltage and still maintain the same performance constraint. Since there is a quadratic relationship in voltage and a near-linear relationship in computational power as we increase the number of cores, the overall energy consumed reduces until the core reaches the near threshold regime. If we lower the voltage further we see less energy gain for a similar decrease in voltage. In this graph the point at which the cores begin running in the near threshold regime is around 4-6 cores. Beyond that point the improvement form lowering the voltage is less than the overhead of making the application more parallel and the total increase in number of cores at a lower voltage.

The L1 energy increases from front to back, decreasing number of clusters. This is because the fewer clusters that are present in the system the larger the L1 caches will become. This occurs because we are holding the total cache size on the chip constant. As we increase the cache size, the energy per access is increased, and therefore the overall L1 energy increases. As we go from left to right, increasing the number of cores per cluster, the energy increases. As we increase the cluster size, we require that the L1 operate at a higher frequency to satisfy the requests. This increase in frequency requires us to increase the voltage of the L1 and the energy savings from the faster communication amongst the cores is outweighed by the increase in the energy per access at the higher voltage.

In Figure 9 we present the L2 cache energy. It is increasing from left to right and from front to back. This increase is related to the larger number of cores. In particular there are more cores sharing the same total L1 space, because we held it constant. This results in more contention and a higher miss rate among the L1's. With more L1 misses there will be more L2 accesses and the energy consumed by the L2 will increase. This increase is highly dependent on how much contention and sharing there is among the different applications.

When the components are summed for the different figures you result in wanting few clusters to reduce the L2 traffic, but more clusters to keep the L1 sizes smaller and less energy hungry. This summation leads to an optimal point of around 2 clusters. The core energy dictates that the optimal total number of cores is somewhere around 4-6 cores. Divided into 2 clusters that means either 2 or 3 cores per cluster. Thus resulting in the energy optimal point of 2 cores per cluster and 2 clusters for this benchmark with 64kB of L1 cache per cluster. This same analysis was done holding either the cluster size or number of clusters constant to understand the effects of the L1 size on the optimal solution. The graph is consistent with the points presented in Section 3.2 and was not presented here for brevity.

6.5. Global Optimal Solutions

Now that we showed the energy optimal point for the Cholesky benchmark we ran the same analysis for additional SPLASH2 benchmarks and the results are presented in Table 2. The table shows the global energy optima for different applications in the SPLASH2 benchmark suite, Clustering is optimal for 6 applications with the optimal cluster size of 2 cores.

6.6. Optimality under Different Performances

So far we have compared the energy savings under constant baseline performance, 233MHz. The optimal V_{dd}/V_{th} configuration changes with target performance, therefore we present the optimal energy consumption for Cholesky under different performance requirements Figure 10(b). The MP scaling uses clustering and optimal V_{dd}/V_{th} selection. Performance on the x-axis refers to the frequency of the baseline single-core single-L1 machine. With reduced target performance, from right to left in Figure 10(b), the energy savings increases first because of relaxed frequency constraints on each core and L1 cache, both of which operate in the near threshold regime. Then energy consumption increases because the cores begin to scale out of the near threshold and into the subthreshold regime causing considerable performance loss in comparison to the power gain. Also the lengthened execution time prolongs L2 leakage energy.

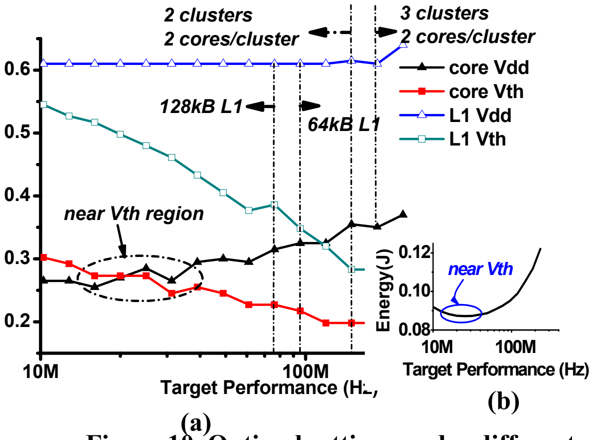
TABLE 2. Each Benchmarks Optimal Config.

	n_c	k	L1 size (kB)*	*energy savings*
cho	3	2	64	70.8%
fft	2	2	32	72.6%
fmm	8	2	128	79.7%
luc	3	2	32	77.8%
lun	2	2	64	68.4%
rad	16	1	128	84.2%
ray	3	2	128	65.1%

k: # of cores per cluster n_c: # of cluster *L1 size is per cluster
energy savings is relative to baseline uniprocessor machine

The V_{dd} and V_{th} settings with different performance targets are shown in Figure 10(a). The V_{dd} for the L1 slightly reduces from 233MHz to 100MHz due to relaxed frequency. However it holds around 600mV because lower V_{dd} implies significant up sizing as described in Section 3. And this area increase from SRAM redesigning nullifies the savings from reduced V_{dd}. The V_{th} increases with lowered performance requirements to balance switching energy and leakage energy. The cores operate at a significantly lower V_{dd} and V_{th} than the L1s for all of the swept performance range because it is running at half the speed of the L1. In addition logic gates are more robust than SRAM and voltage scale without area overhead.

In Figure 10(a) the optimal number of clusters increases from 2 to 3 for targets above 150MHz. This can be attributed to the fact that in order to meet the



(a) (b)

Figure 10. Optimal settings under different target performances

higher performance constraint we need more computational power. The increase in the number of clusters, and thus total cores, requires less energy than voltage scaling the smaller number of cores to meet the same constraint. Figure 10(a) also shows that the optimal L1 size changes from 64kB to 128kB for targets below 76MHz because the relative contribution of L2 energy consumption starts to increase. A larger L1 helps to suppress L2 accesses. Although the larger L1 incurs more energy per access, the amount of energy saved from reducing the L2 accesses outweighs any increase in the L1.

Finally, we have highlighted the near V_{th} region on both plots in Figure 10. Optimal energy consumption is achieved at this voltage regime and at a target frequency of tens of megahertz (~15MHz-50MHz). These operating frequencies are much higher than those shown in other subthreshold work[1] and are well suited for parallelizable embedded applications requiring higher performance, but where battery life is important. Such applications might include MPEG and MP3 decoding.

7. Split L1 Cache

We further investigated the impact of split instruction (IL1) and data (DL1) caches. By splitting the unified L1 cache we present a few more architectural choices. First we could cluster both the IL1 and DL1 for each cluster as in Figure 11(a). Second we could explore other architectures where we only cluster the IL1 and leave each core a private DL1 as in Figure 11(b), or vice versa as in Figure 11(c). In this section we will explore those three different design choices and discuss their impact on different programing models.

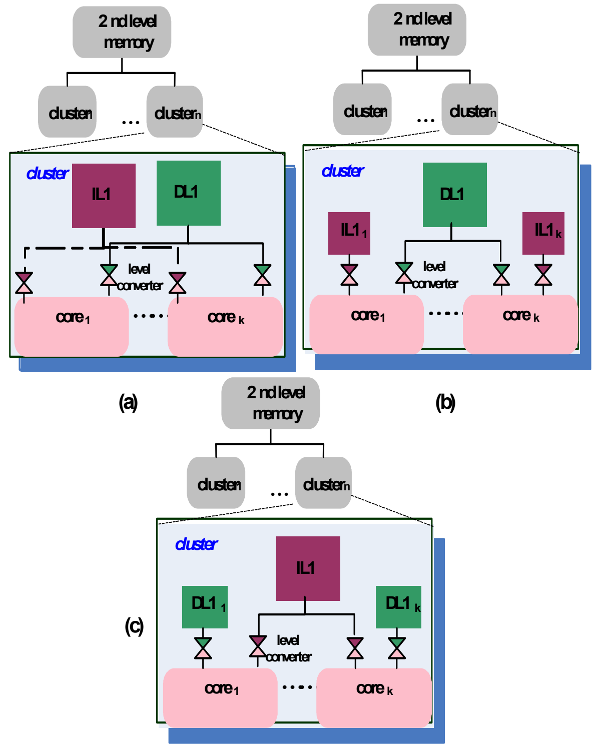


Figure 11. (a) Clustered I and D (b) Clustered D and (c) Clustered I architectures

7.1. Architectural Trade-offs

There are several architectural trade-offs that occur when we consider splitting the unified cache. By splitting the L1 cache we remove any contention that was occurring between the instruction and data streams. In some benchmarks the data stream constantly evicted instructions from the cache do to conflicts and the overall number of cycles to complete the program was increased. By removing these conflicts we can decrease the number of cycles improving energy efficiency. In the next two subsections we break down the benefits of clustered versus private versions of both the IL1 and DL1.

7.1.1. Clustered IL1 vs. Private IL1

At first glance clustering the instruction cache seems to make sense for applications where each thread is executing the same instructions, i.e. SIMD. The reasoning is that by sharing the cache you can reduce the die size and improve performance because only one thread will incur a miss, and others will hit the line in the shared cache reducing execution time and thus energy. But if the instruction stream is small, the instruction cache can end up with a low miss rate and a high access rate. Since we are limited by Amdahl's law in only improving performance for misses, the gains we see in this case are marginal. Further, by clustering the instruction cache we require the cache to operate at a higher frequency, and therefore a higher energy per access. However, if the instruction miss rate is large enough, then it begins to make sense to share the instruction cache to improve the access latencies. In applications that run different sets of instructions on each thread, i.e. MIMD or mutliprogrammed workloads, it does not makes sense to cluster the instruction cache.

7.1.2. Clustered DL1 vs. Private DL1

Unlike the instruction cache, the data cache is only accessed by around 30% of the instructions in the SPLASH2 benchmark suite[6]. With the lower access rate, higher miss rate, and larger data sharing, the data cache is suited well for clustering. If the application does large amounts of communication through shared memory or shares large amounts of data, clustering the data cache will result in shorter synchronization delays and access times. This results in the program completing in fewer cycles leading to energy savings. On the other hand, clustering the data cache does not make sense if the applications do not share data or synchronization variables because it will lead to contention in the cache and no improvement for the core in terms of cycles to completion.

7.2. Benchmarks

We chose three different SPLASH2 benchmarks to analyze on the three different architectures. We wanted to get a variety of different synchronization schemes. We picked Cholesky because of the high use of locks and pauses but low use of barriers. We chose LU because it uses many barriers and no locks or pauses. And finally we chose FFT which uses very few barriers, locks, or pauses.

7.3. Simulation

We simulated the benchmarks with the same configuration choices as before, but now we allowed the L1 cache to be split. We varied the IL1 and DL1 sizes from 4-128kB independently while exploring all 3 configurations in Figure 11.

7.4. Results

The results of the optimal configuration for each of the three architectures on the three benchmarks is presented in Table 3. The energy savings is presented in percentage reduction over the baseline single core machine with 64kB split IL1 and DL1s. You can see that the benchmarks prefer to run with a clustered data cache and a private instruction cache for the best overall energy savings.

TABLE 3. Optimal Configurations Split L1

	IL1	DL1	n_c	k	IL1 Size	Dl1 Size	Energy Savings
cho	Unified		3	2	64kB Unified		70.8%
	Clustered	Private	3	2	8kB	128kB	73.0%
	Private	Clustered	2	2	16kB	64kB	76.5%
	Clustered	Clustered	3	2	16kB	64kB	71.1%
lun	Unified		2	2	64kB Unified		68.4%
	Clustered	Private	2	2	8kB	32kB	64.2%
	Private	Clustered	1	2	16kB	32kB	72.9%
	Clustered	Clustered	2	2	8kB	64kB	66.4%
fft	Unified		2	2	32kB Unified		72.6%
	Clustered	Private	1	2	4kB	64kB	72.1%
	Private	Clustered	1	2	8kB	64kB	75.0%
	Clustered	Clustered	2	2	4kB	32kB	69.7%

The SPLASH2 benchmarks all exhibited low miss rates for small instruction caches (4-16kB) so it did not make sense to cluster the instruction cache as described in Section 7.1.1. This is easily seen in both the FFT and LU benchmarks where each instance of clustering the IL1 results in worse performance than the unified version. In the case of Cholesky, the increase seen over the unified version when the IL1 is clustered is a result of less contention on cache lines between the instruction and data caches. With less contention we reduce the number of cycles and gain more energy savings. Even though for Cholesky a clustered IL1 outperforms the unified cache case, a clustered DL1 and private IL1 is still the overall optimal choice. This happens for the same reasons as the FFT and LU benchmarks.

8. Related Work

There has been a myriad of different work done in both the fields of subthreshold design and parallel architectures for low power. The most recent work done for power aware parallel architectures was by Li et al. [21]. Li proposes a dynamic runtime method to use dynamic voltage and frequency scaling available on cores to optimize power given a performance constraint on parallel applications. Our work differs in that we focus on near threshold operation and the additional architectural choices for clustering that it allows. Li's work could be extended to work in combination with our design to dynamically put nodes to sleep to find the most optimal power configuration available. It could also be used to allow only one core per cluster to operate when applications require more cache space, or to dynamically reassign threads to cores within clusters that communicate more often.

Other power aware parallel CMP and simultaneous multithreading (SMT) designs have been proposed by [22,23,24,25,26,27]. This work differs from both Li's [21] and our work in that it focuses on mutliprogrammed workloads, whereas our work focuses on parallel applications.

Huh et al. [29] does an in depth study of the design space of CMP's. However they do not evaluate either power considerations or the possibility of clustering multiple cores to the same L1 cache. Other work exploring the design space of CMP's that considers power budgets was done by Ekman and Stenstrom [30] although their work focuses on the types of cores that should be used in a CMP given the amount of parallelism applications present. Our work differs in that we assume a fixed core design and chose the optimal number of cores and the amount of clustering that occurs.

There is also an extensive body of work on reducing energy overheads in bus based CMP's [17,18,19,20,28]. Moshovos [17] and others provide different techniques to reduce the number of snoops into L1 caches by filtering out requests that will not find the block in that cache. This technique could be used to lower the energy consumption of the snoops in our system removing one of the performance benefits of clustering. However, clustering not only eliminates snoop energy but also helps reduce the latency to shared data structures in the cache within a cluster. This reduction in latency means the system can be run at a slower frequency and still complete in time. Snoop filtering techniques help to reduce the overall system energy further but when removing the energy consumed by snoops in our simulations we still find clustering to be the optimal choice.

9. Conclusions

In this paper we have investigated the optimal threshold voltage selection for energy efficient near threshold design. By separately controlling the supply voltage and the threshold voltage of the core and the memory, we can achieve better energy efficiency. We have combined these techniques with a novel multiprocessor architecture where multiple cores share one faster L1 cache in a cluster to further improve energy savings. We found that for a typical SPLASH2 application the proposed architecture can provide about 70% energy savings over a uniprocessor system and about 53% over conventional multiprocessor scaling. The optimal cluster size is 2 cores for most of the SPLASH2 benchmarks that we investigated, and the system achieves best energy efficiency when operating in the near threshold voltage regime. The optimal target frequency was that of a single 10-50MHz core, showing that this technique is well suited for parallelizable embedded applications requiring higher performance, but where battery life is important such as MPEG decoding. We further studied the impact of splitting the instruction and data cache. We found that clustering the data cache while keeping a separate private instruction cache per core performed the best for the SPLASH2 applications providing an energy savings of up to a 77% reduction over that of a single core machine.

Acknowledgements

The authors acknowledge the support of NSF, SRC, Intel, and the Gigascale Systems Research Focus Center, one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program.

References

[1] B. Zhai, L. Nazhandali, *et al.*, "A 2.60pJ/Inst Subthreshold Sensor Processor for Optimal Energy Efficiency", *IEEE VLSI Technology and Circuits*, 2006

[2] B. Zhai, R. Dreslinski, *et al.*, "Energy Efficient Near-threshold Chip Multi-processing", *ISLPED*, 2007

[3] A. Wang, A. Chandrakasan, "A 180mV FFT processor using subthreshold circuits techniques", *IEEE ISSCC* 2004

[4] B. Zhai, D. Blaauw, *et al.,* "Theoretical and practical limits of dynamic voltage scaling", *DAC* 2004

[5] B. Calhoun, A. Chandrakasan, "Characterizing and modeling minimum energy operation for subthreshold circuits," *ISLPED 2004*

[6] S. C. Woo, M. Ohara, *et. al.* "The SPLASH-2 Programs: Characterization and Methodological Considerations", *ISCA,* 1995.

[7] T. Sakurai and A. Newton, "Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas," *IEEE JSSC*, vol. 25, no. 2, pp. 584-594, Apr. 1990.

[8] http://www.arm.com/products/CPUs

[9] B. Zhai, S. Hanson, *et al.*, "Analysis and Mitigation of Variability in Subthreshold Design", *IEEE ISLPED*, 2005

[10] B. Calhoun and A. Chandrakasan, "A 256kb Sub-threshold SRAM in 65nm CMOS", *IEEE ISSCC*, 2006

[11] N. Verma, A. Chandrakasan, "A 65nm 8T Sub-Vt SRAM Employing Sense-Amplifier Redundancy", *IEEE ISSCC*, 2007

[12] T-H. Kim, J. Liu, *et al.*, "A High-Density Subthreshold SRAM with Data-Independent Bitline Leakage and Virtual-Ground Replica Scheme", *IEEE ISSCC*, 2007

[13] B. Zhai, D. Blaauw, *et al.*, "A Sub-200mV 6T SRAM in 0.13um CMOS", *IEEE ISSCC*, 2007

[14] M. J. M. Pelgrom, *et al.*, "Matching properties of MOS transistors," *IEEE JSSC*, vol. 24, no. 5, pp. 1433-1440, 1989.

[15] N. L. Binkert, R. G. Dreslinski, *et al.*, "The M5 Simulator: Modeling Networked Systems.", *IEEE Micro*, pp. 52-60, 2006

[16] N. S. Kim, K. Flautner, *et al.* "Single-Vdd and Single-Vt Super-Drowsy Techniques for Low-Leakage High-Performance Instruction Caches", *IEEE/ACM ISLPED*, 2004.

[17] A. Moshovos. "RegionScout: Exploiting coarse grain sharing in snoop-based coherence", *ISCA*, 2005.

[18] A. Mosohovos, G. Memik, et al. "Jetty: Filtering snoops for reduced energy consumption in SMP servers". *HPCA*, 2001.

[19] C. Chung, J. Kim, et al. "Reducing snoop-energy in shared bus-based MPSoCs by filtering useless broadcasts". *GLSVLSI* '07.

[20] M. Ekman, F. Dahlgren, et al. "Evaluation of snoop-energy reduction techniques for chip-multiprocessors", *WDDD*, 2002.

[21] J. Li, F. Martinez. "Dynamic power-performance adaptation of parallel computation on chip multiprocessors". *HPCA*, 2006

[22] J. Donald, M. Martonosi. "Temperature-aware design issues for SMT and CMP architectures". *WCED*, 2004.

[23] S. Ghiasi, D. Grunwald. "Design choices for thermal control in dual-core processors". *WCED*, 2004.

[24] R. Kumar, K. Farkas, et al. "Single-ISA heterogenous multi-core architectures: The potential for processor power reduction". *MICRO*, 2003.

[25] Y. Li, D. Brooks, et al. "Performance, energy, and temperature considerations for SMT and CMP architectures". *HPCA*, 2005.

[26] R. Sasanka, S. Adve, et al. "Comparing the energy efficiency of CMP and SMT architectures for multimedia workloads. *ICS* 04.

[27] J. Send, D. Tullsen, et al. "Power-sensitive multithreaded architecture". *ICCD*, 2000.

[28] C. Saldanha, M. Lipasti. "Power efficient cache coherence". *WMPI*, 2001.

[29] J. Huh, D. Burger, et al. "Exploring the design space of future CMP's". *PACT*, 2001.

[30] M. Ekman, P. Stenstrom. "Performance and power impact of issue-width in chip-multiprocessor cores". *ICPP*, 2003.