Reconfigurable Multicore Server Processors for Low Power Operation

Ronald G. Dreslinski, David Fick, David Blaauw, Dennis Sylvester, Trevor Mudge

University of Michigan, Advanced Computer Architecture Labratory,
2260 Hayword, Ann Arbor, MI, 48109
{rdreslin,dfick,blaauw,dennis,tnm}@eecs.umich.edu

Abstract. With power becoming a key design constraint, particularly in server machines, emerging architectures need to leverage reconfigurable techniques to provide an energy optimal system. The need for a single chip solution to fit all needs in a warehouse sized server is important for designers. This allows for simpler design, ease of programmability, and part reuse in all segments of the server. A reconfigurable design would allow a single chip to operate efficiently in all aspects of a server providing both single thread performance for tasks requiring it, and efficient parallel processing helping to reduce power consumption. In this paper we explore the possibility of a reconfigurable server part and discuss the benefits and open questions still surrounding these techniques.

Keywords: Reconfigurable, Low Power, Server Architectures.

1 Introduction

The exponential growth of the web has yielded an equally dramatic increase in the demand for server style computers. According to IDC the installed base of servers will exceed 40 million by 2010. In fact, these figures may be conservative as there is a continual flow of unanticipated applications coming online. For example, Facebook, which is only 3 years old, is expected to grow from 1,000 to 10,000 servers in a year. The growth in servers has been accompanied by an equally dramatic growth in the demand for energy to power them. Furthermore, the cost of this power and its associated cooling is approaching the cost of the servers themselves [1]. For example, it is estimated that the five largest internet sites consume at least 5MWh of power each [2]. Meanwhile, through technology advancements and new design techniques such as 3D die stacking, Moore's law continues to hold. This means there is an increasing number of transistors available on a chip. However the power allocated for those transistors remains constant. This power envelope means that either new, low power architectures need to be developed, or the additional transistors supplied by Moore's law will go unutilized.

At the same time the need for a chip that satisfies all applications within a server environment is critical. Designers prefer to use the same chip for all portions of the server to ease the programming constraints as well as keep cost and maintenance to a minimum. Simply creating a low power chip capable of handling only a portion of the workloads required in a server will not be economically viable unless current designers rework their approach.

Therefore, there is a need for a system that supports both low power throughput computing and fast single threaded performance to address the coming problems in large scale servers. To address this we propose a system leveraging near-threshold voltage scaling and parallel processing to reduce the power consumption of the throughput oriented applications in a server. At the same time we employ reconfigurable techniques to adapt to response times of the throughput computing or to handle applications where single threaded performance is critical. This reconfigurability will also rely heavily on the ability of the OS to measure and adapt the chip to reduce power consumption while still maintaining the needed performance. In this paper we explore an example architecture and some of the difficulties associated with the OS scheduling. We present some early solutions and point to future research directions for remaining problems.

2 Reconfigurable Architecture

Although the techniques discussed in this paper are not directly tied to a particular architecture, the following processor description will serve as an example for illustration throughout the rest of the paper. In Figure 1, we present our example reconfigurable architecture, which has several interesting design points that we will discuss in the following subsections. The basic design is a machine with 66 cores. Each core is tied to the same ISA so that code can be migrated freely between cores without concern for the ability of any given core to complete the task.

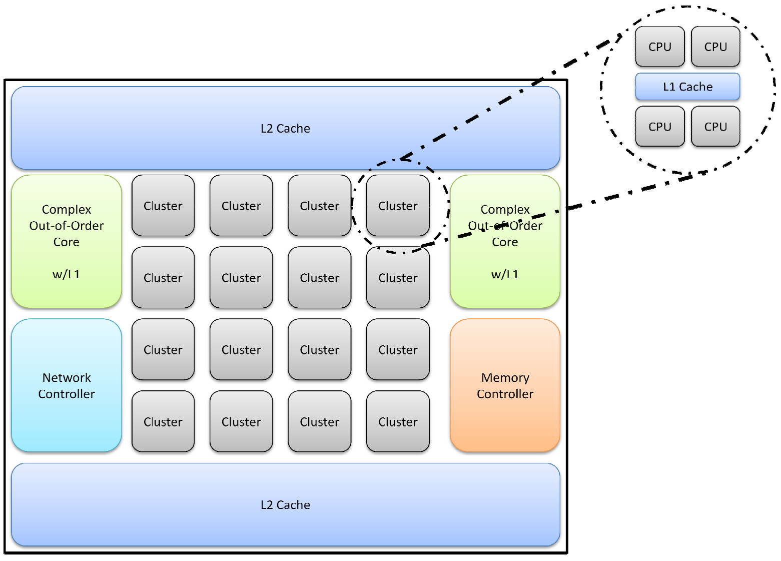

Fig. 1. Example reconfigurable server architecture, using clustering techniques.

2.1 Core Types

There are two basic core types in the design. The first core type is a simple, in-order execution core. This core is replicated 64 times across the architecture and is to be used in highly parallel tasks to reduce energy consumption. In figure 1 it is the core replicated 4 times in each cluster. The core itself is designed to run at a range of frequency/voltage pairs. Depending on computational demands the cores can be scaled to offer faster performance, but with increased power consumption. At the lowest end of operation the cores operate in what we term the near-threshold (NTC) operating region[3]. This region is where the supply voltage is at or just above the threshold voltage of the transistor. In this region cores see approximately a 100x reduction in power with only a 10x reduction in performance, resulting in a 10x reduction in energy. The processor is not operated at subthreshold [4,5] voltage levels, due to the poor energy/performance tradeoff that occurs at this point. See figure 2 for a view of the tradeoffs of delay and energy in different supply voltage regions.

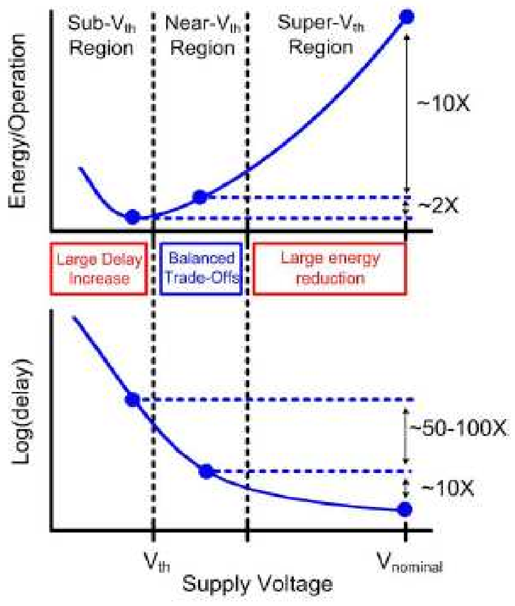

Fig. 2. Energy and delay for different supply voltage operating regions.

The second core type is a more complex out-of-order core. This core is designed to operate only at full voltage and is used to perform single threaded tasks that cannot be parallelized, or time critical tasks that would take too long on the simple cores. In a server farm many tasks still require single thread performance and any single solution chip needs to be able to provide this performance in addition to any energy saving parallel cores it offers. These cores can either be enabled, or power gated when not needed to reduce energy. Depending on the thermal characteristics of these cores, nearby simple core clusters may need to be disabled while operating these cores.

2.2 Clustered Architecture

In the work done by Zhai et. al [3,6], they propose the use of parallelism in conjunction with NTC operation to achieve an energy efficient system. While traditional superthreshold many-core solutions have been studied, the NTC domain presents unique challenges and opportunities for architects. Of particular impact is the reliability of NTC memory cells and differing energy optimal points for logic and memory, as discussed below.

Zhai's work showed that SRAMs, commonly used for caches, have a higher energy optimal operating voltage than processors, by approximately 100mV [3]. This results from the lower activity in caches, which amplifies leakage effects. SRAM designs also face reliability issues in the NTC regime, leading to a need for larger SRAM cells or error correction methods, further increasing leakage and the energy optimal operating voltage. Due to this higher optimal operating voltage, SRAMs remain energy efficient at higher supply voltages, and thus at higher speeds, compared to logic. Hence, there is the unique opportunity in the NTC regime to run caches faster than processors for energy efficiency, which naturally leads to architectures where multiple processors share the same first level cache.

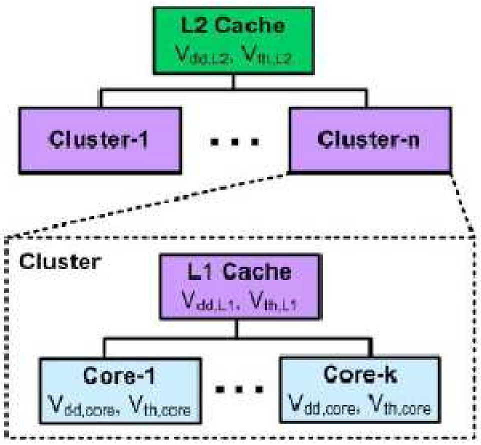

Fig. 3. Cluster Based Architecture.

These ideas suggest that we create an architecture with *n* clusters, each with k cores, where each cluster shares a first level cache that runs *k* times faster than the cores (Figure 3). Different voltage regions are presented in different colors and use level converters at the interfaces. This architecture results in several interesting tradeoffs. First, applications that share data and communicate through memory, such as certain classes of scientific computing, can avoid coherence messages to other cores in the same cluster. This reduces energy from memory coherence. However, the cores in a cluster compete for cache space and incur more conflict misses, which may in turn increase energy use. This situation can be common in high performance applications where threads work on independent data. However, these workloads often execute the same instruction sequences, allowing opportunity for savings with a clustered instruction cache. Initial research of this architecture [3,6] shows that with a few processors (6-12), a gain of 5-6X performance improvement can be achieved.

Since the caches are operating at a frequency that is higher than the cores, it is possible to turn off some of the cores in the cluster, clock the remaining cores at a higher frequency, and not have to change the cache frequency. This is beneficial because the SRAM needs to be validated at each operating frequency and the timing of signals, particularly relating to the sense amplifier, are sensitive to timing changes, whereas core logic timing scales predictably with voltage. In our example architecture we present a system with clusters of size 4, meaning there are 4 cores in the cluster and the cache is operated at 4x the frequency of the cores. We propose to allow each cluster to be operated in 4 modes. These modes correspond to the number of cores being operated at a particular time, while the remaining cores are power gated off. Figure 4 shows how each of the 4 configurations would look, assuming a 75MHz NTC operating point when all 4 cores are enabled. The benefit of using these modes is that the system is able to trade-off power for response time. So if the system is given a response time constraint, the OS can adapt the number of cores and frequency to achieve the energy optimal solution. Figure 5 shows a simulation run using the M5 [7] full system simulator for a cluster of size 4. The benchmark being run is a simplified version of the SpecWeb[8] benchmark, in a system with a 64kB clustered ICache and a 256kB clustered DCache. As can be seen the throughput in the system remains nearly constant, but the response time of the system can be reduced at the expense of additional power.

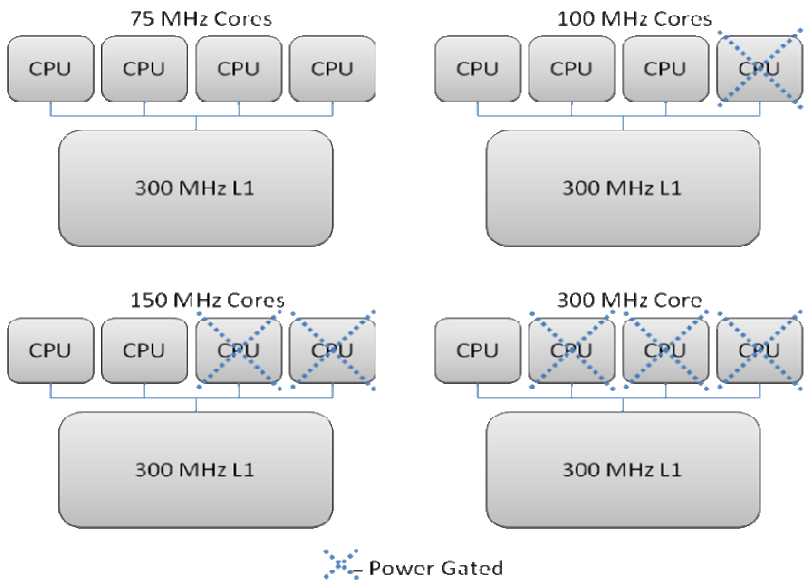

Fig. 4. Modes of operating a 4 core cluster with 75MHz NTC frequency.

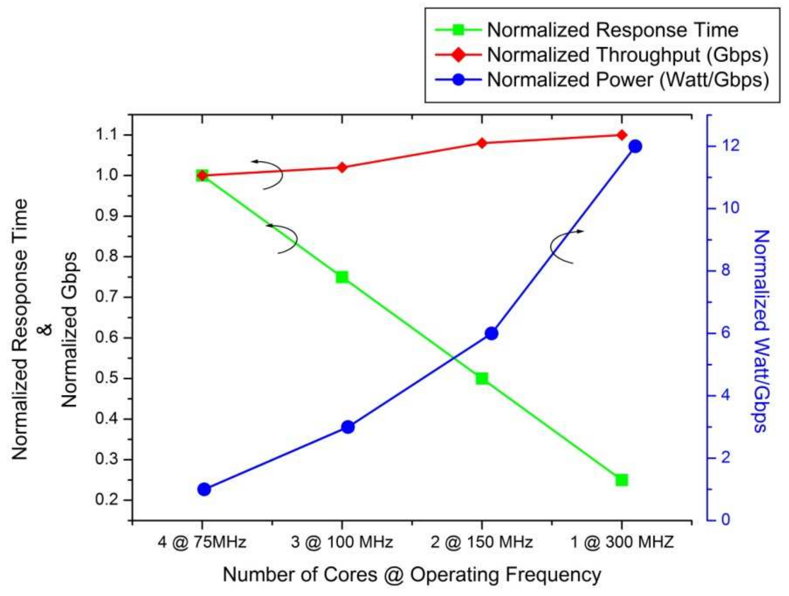

Fig. 5. Tradeoff of response time and power for a cluster based architecture.

3 Thread to Core Mappings and Thread Migration

To fully utilize the capabilities of the reconfigurable system the operating system will need to carefully orchestrate the mappings and migrations of threads to cores. The system will need to have a set of constraints in terms of performance desired and the operating system will perform some heuristic mapping and migration schemes to optimize the system for power consumption. In the following subsections we detail some of the decisions the operating system will need to make and present some examples of performance metrics that can be used to guide these decisions.

3.1 Large Data Cache Requirement

Since the cores within a cluster share the same L1 cache space, there is the potential for threads running on cores within the same cluster to thrash each others data. In order to prevent this from impacting the overall runtime, the OS will need to detect these competing threads and migrate them to different clusters. Preferably this will be done by collocating the threads with large data cache requirements with threads having small cache demands or on clusters where fewer cores are enabled reducing the cache pressure. One possible detection scheme for this condition would be a performance counter that tracks the number of cache evictions that a particular thread causes of data that belonged to a different thread. This technique would employ 2-

bits of overhead on each cache line to distinguish the core for which the cache line was originally fetched. On an eviction, if the core is not the one who originally fetched the data, a counter is incremented for the core causing the eviction. When the OS reads these counters it will be able to get an approximation for the negative impact a particular thread has on others in the same cluster, allowing a decision to be made about thread migration.

3.2 Shared Instruction Stream/Data

In some cases threads can either run the same instruction streams, i.e. SIMD, or may operate on the same pieces of shared data. In these cases it is beneficial for the OS to map these threads to the same cluster. There are several advantages to doing so. First, there is less cluster based evictions due to the sharing of the cache line. Second, in the case of data it avoids the costly process of moving data around when multiple cores wish to modify the data. Third, the threads act as prefetchers for each other reducing the latency of the system. As in the previous case, section 3.1, the same 2-bit field can be added to each cache line noting the core in the cluster that fetched the cache line. 10 counters can be used to keep track of pairs of cores that shared a line. If a cache line is ever read by a core different than the one that fetched the data the corresponding counter is incremented. This provides the OS with a count of currently scheduled threads in a cluster and the amount of sharing that is taking place. To detect threads that are not currently in the same cluster that would benefit from being in the same cluster a more complex scheme would need to be designed on top of the coherence protocol to detect these conditions. This is left for future work.

3.3 Producer/Consumer Communication Patterns

In some programming models there are producer/consumer data relationships. This is where one threads output is constantly the input to another thread. In these types of patterns it is beneficial for the OS to migrate the threads to the same cluster. By doing so, the consumer can avoid going out of the cluster to get the data produced by the producer. Depending on the interconnect in the system, if it isn’t possible to put them in the same cluster, then there is also benefit by putting them in clusters near each other. For example in a network-on-chip style interconnect having them close reduces the number of cycles it takes to transfer the data, and it relieves congestion on the network. An elegant solution to detecting these communication patterns has not yet been worked out, but a possible solution in a directory based coherence machine might involve tracking some read/write patterns on cache lines at the home node. This again is left for future work.

3.4 Single Thread Performance

Some threads will require more performance because they may lie on the critical path of execution. The OS needs to identify these threads and migrate them either to clusters with fewer cores enabled, and thus a faster frequency, or in the extreme case to one of the complex out-of-order cores in the system. These threads can hopefully be identified by either the programmer or the compiler, but in some case may rely on hardware feedback. In most cases these threads tend to serialize the operation of the machine, so in situations where few threads are running, identifying the ones that have been running the longest may indicate which threads need to be migrated to faster cores. The OS may need to migrate threads to the complex cores for short periods of time and measure overall utilization to determine if there is a positive impact of running the threads at these locations.

4 Conclusions

With power becoming a major concern, particularly in servers, new energy optimal architectures need to be explored. In this paper we looked at the use of reconfigurable architectures to provide a single chip solution to a broad range of targets. When parallelism is abundant the system can adapt and save large amounts of energy, at the same time when single thread performance is the bottle neck the system can be reconfigured to provide the necessary throughput. The architecture explored in this paper employed near-threshold techniques, L1 cache clustering, and heterogeneous design (in-order and out-of-order cores) to achieve extremely energy efficient computation. The paper also looked forward at the difficult task the OS will have with managing the thread to core mapping for energy optimality, and proposed some initial techniques that might be employed by the OS.

References

1. Lim, K., Ranganathan, P., Chang, J., Patel, C., Mudge, T., Reinhardt, S.: Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments. In Proceedings of the 35th ISCA, 315-326 (2008)
2. Report to Congress on Server and Data Center Energy Efficiency, US Environmental ProtectionAgency,http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf.
3. Zhai, B., Dreslinski, R.G., Mudge, T. Blaauw, D., Sylvester, D.: Energy efficient near-threshold chip multi-processing. ACM/IEEE ISLPED, pp. 32-37 (2007).
4. Zhai, B., Blaauw, D., Sylvester, D., Flautner, K.: Theoretical and practical limits of dynamic voltage scaling. ACM/IEEE Design Automation Conference, pp. 868-873 (2004).
5. Wang, A., Chandrakasan, A.: A 180mV FFT processor using subthreshold circuit techniques. IEEE International Solid-State Circuits Conference, pp. 292-529 (2004).
6. Dreslinkski, R. G., Zhai, B., Mudge, T., Blaauw, D., and Sylvester, D.: An Energy Efficient Parallel Architecture Using Near Threshold Operation. In Proceedings of the 16th PACT, 175-188 (2007).
7. Binkert, N.L., Dreslinski, R.G., Hsu, L.R., Lim, K.T., Saidi, A.G., Reinhardt, S. K.: The M5 Simulator: Modeling Networked Systems. IEEE Micro, vol. 26, no. 4, pp. 52-60, July/August (2006).
8. SpecWeb99 benchmark. http://www.spec.org/web99.