The KILL Rule for Multicore

Anant Agarwal
MIT and Tilera Corp.

Markus Levy
EEMBC

ABSTRACT

Multicore has shown significant performance and power advantages over single cores in commercial systems with a 2-4 cores. Applying a corollary of Moore's Law for multicore, we expect to see 1K multicore chips within a decade. 1K multicore systems introduce significant architectural challenges. One of these is the power efficiency challenge. Today's cores consume 10's of watts. Even at about one watt per core, a 1K-core chip would need to dissipate 1K watts! This paper discusses the "Kill rule for multicore" for power-efficient multicore design, an approach inspired by the "Kiss rule for RISC processor design". Kill stands for *Kill if less than linear,* and represents a design approach in which any additional area allocated to a resource within a core, such as a cache, is carefully traded off against using the area for additional cores. The Kill Rule states that we must increase resource size (for example, cache size) only if for every 1% increase in core area there is at least a 1% increase in core performance.

Categories and Subject Descriptors
C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors) – *parallel processors.*

General Terms
Measurement, Performance, Design, Economics, Experimentation, Theory

Keywords
Kill Rule, multicore, core, computer architecture, power efficiency, parallel computing, CMP, stream processing, tiled multicore

1. INTRODUCTION

Application demand for computing cycles in virtually every domain, from the embedded market to the desktop PC, continues to increase unabatedly. Modern video workloads, for example, require 10 to 100 times more compute power than that of a few years ago due to increasing resolutions (from SD to HD), more sophisticated compression algorithms (MPEG2 to H.264), and greater numbers of channels. Unfortunately, the delivered performance of conventional sequential processors and digital signal processors has not kept pace with this demand. In fact, the performance of sequential processors has tapered off even as the number of transistors that is available on a single chip has continued to increase exponentially. This phenomenon is called *Moore's Gap.* The reasons for this widening gap between the growing number of transistors on a single chip and its delivered performance include diminishing returns from single processor mechanisms such as caching and pipelining, wire delays, and power envelopes.

Multicore architectures help close the Moore's Gap. Multicore refers to a single chip containing multiple visibly distinct processing engines, each with independent control (or program counters), as illustrated in Figure 1. Multicore uses the multiple-instruction-multiple-data (MIMD) style of computation. Multiple voltage-scaled lower-frequency cores on a single chip offer significantly more performance and power efficiency than single cores, provided the applications have parallelism. Indeed, contemporary applications and workloads of today have ample parallelism. In addition to the video example given at the beginning of the article, plenty of other applications exist that demonstrate parallelism properties. These include networking (e.g. IP forwarding), throughput oriented servers, wireless (e.g. Viterbi decode, FIR filters), security firewalls (e.g. AES), automotive (engine control), and many others.

If the cores use the same number of transistors, more cores can be added in a multicore chip as the number of available transistors increases. In fact, using a corollary of Moore's Law, we can say that the number of cores on a chip will double every 18 or 24 months. Given that dual and quad cores from leading commercial vendors are in widespread use today, and research prototypes of

16-core multicores were available in university in 2002[1], it is highly likely that we will see 1000-core multicores in the early part of the next decade.

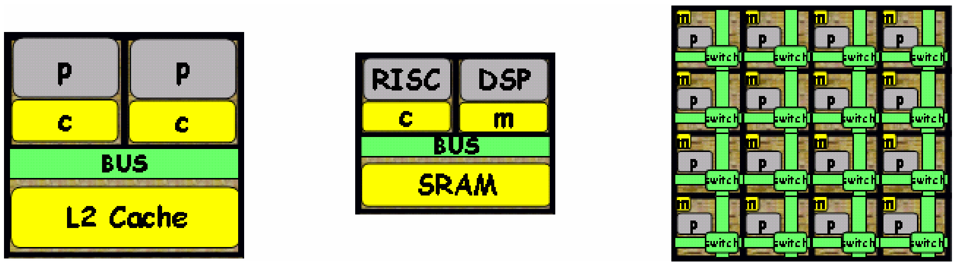

Figure 1 Three examples of multicore implementations, a bus-based multicore showing two identical CPUs, one with a RISC CPU and a DSP, and a tiled multicore architecture with 16 identical CPUs

Multicore chips with large numbers of cores introduce significant architectural and programming challenges. One of the architectural challenges is power efficiency. Today's cores run at 10's of watts. Even at about one watt per core, a 1K core chip would need to dissipate a staggering 1K watts! For a fixed frequency, power relates to chip area, so to maximize power efficiency (or performance per watt) it is critical to obtain the most performance out of a given area of silicon. This paper discusses a simple approach for designing the right balance of resources in a multicore to obtain the most out of a given amount of silicon area.

2. KILL RULE FOR MULTICORE

In the days of single core designs, architects would increase cache sizes, clock frequencies, pipeline depths or register file ports, as more area became available. Multicores provide the further choice of including additional cores. When parallelism exists, adding an additional core increases performance proportionally. However, increasing the size of any resource within a core might not increase the performance by the same amount. Thus, given more area, increasing the number of cores and keeping resource sizes relatively small, might result in more performance than increasing resource sizes and keeping the number of cores constant. This new dimension for improving performance and power efficiency in multicore requires us to rethink processor architecture.

The Kill Rule is a simple scientific way of making the tradeoff between increasing the number of cores or increasing the core size. *The Kill Rule states that a resource in a core must be increased in area only if the core's performance improvement is at least proportional to the core's area increase. Put another way, increase resource size only if for every 1% increase in core area there is at least a 1% increase in core performance.*

The rationale for the Kill Rule is that a multicore's performance can always be increased proportionally by adding more cores (assuming application parallelism exists). Thus, increase in a core's size can only be justified if it results in a proportional increase in performance.

Figure 2 illustrates an example applying this rule to choosing an optimal cache size for each core in a multicore chip, using MPEG2 decode as an application example. Suppose we start with a baseline multicore chip design containing 100 cores, each with a 512 byte data cache. Assume that the 512 byte cache occupies 1% of the core's area. Since power relates to area, we will use the kill rule to find the design (in our case, the data cache size and resulting number of cores) that yields the highest performance keeping the area constant to that of 100 cores with a 512 byte data cache.

We will use instructions per clock (IPC) as our metric of performance. IPC is the effective number of instructions that the core or chip can complete per clock cycle. Assume that a third of all instructions are loads or stores. Assume further that the data cache miss rate is "m", and the latency of satisfying a cache miss is 300 cycles. In other words, "m" is the probability that a load or store instruction suffers a cache miss. If each instruction executes ideally (when no cache misses occur) in a core in a single cycle, the IPC is given by

$$\text{Core IPC} = 1/(1 + 0.33 \times m \times 300) = 1/(1 + m \times 100)$$

For MPEG2 decode, in the baseline design with the 512 byte cache, the data cache miss rate is measured to be 25%. Thus the core IPC for the baseline design is 0.04. Since the multicore chip has 100 cores, the aggregate IPC for the entire chip is therefore 4.

If the data cache per core is increased from 512 bytes to 2Kbytes, the resulting area occupied by the data cache is 4%. Only 97 cores will now fit in the same area. The miss rate of the caches decreases to 5%, the IPC for each core increases to approximately 0.17, and so the IPC for the entire chip with the 2Kbyte data cache becomes 0.17 x 97, which is approximately 17.

Notice that increasing the cache size from 512 bytes to 2Kbytes is a good trade by the kill rule since the area of a core increases by 3% (see Figure 3) while the performance of a core increases by 325%.

[1] "**Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams**," by Michael Bedford Taylor, Walter Lee, Jason Miller, David Wentzlaff, Ian Bratt, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jason Kim, James Psota, Arvind Saraf, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. *Proceedings of the International Symposium on Computer Architecture (ISCA),* June 2004.

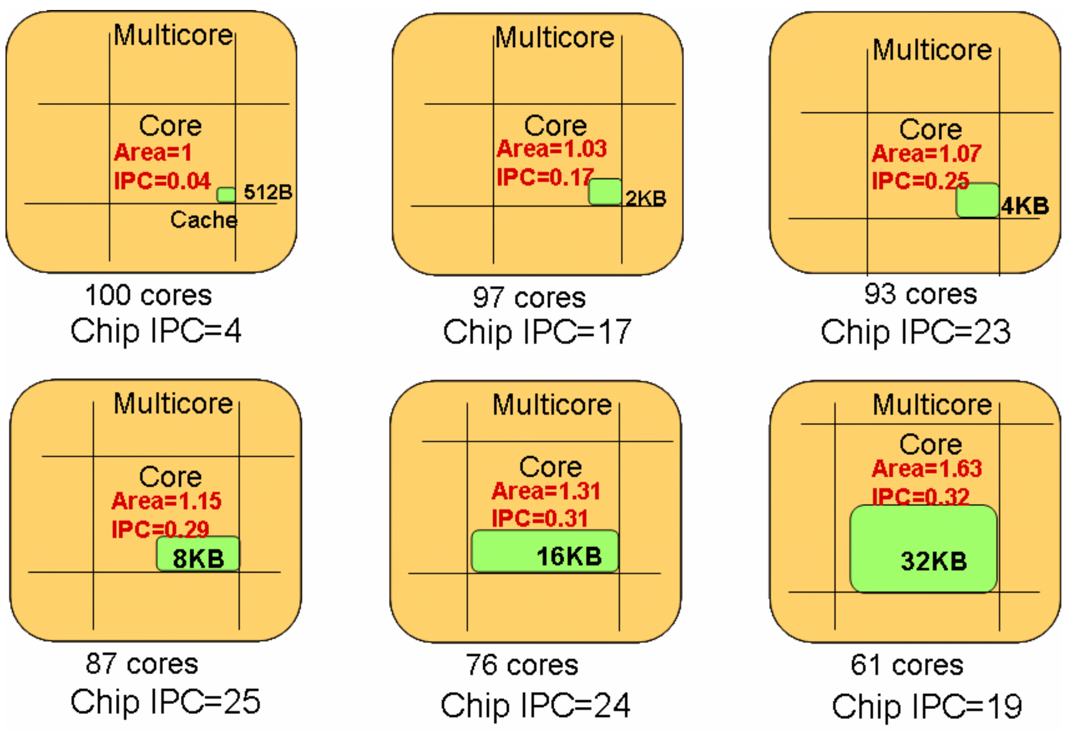

Figure 2 Multicore example illustrating the application of the Kill Rule

Now, suppose we consider doubling the cache size to 4Kbytes. This cache occupies 8% of the chip area. The miss rate decreases further to 3%, the number of cores decreases to 93, and the IPC of each core increases to 0.25.

As depicted in Figure 3, there is a 4% increase in the area of a core, which results in a 47% increase in core performance over the 2Kbyte cache. By the kill rule, increasing cache size to 4Kbytes is a good tradeoff, because the 47% performance increase is greater than the 4% area increase over the 2Kbyte cache.

This trend continues until we try to implement a cache size greater than 8Kbytes. To go from 8Kbytes to 16Kbytes, notice from Figure 3 that an additional 14% of the chip area must be devoted to data cache, while the resulting increase in the core's performance is only 7%. This is a bad tradeoff, since the performance increase of 7% is less than the 14% area increase. Put another way, a 16Kbyte cache for MPEG2 decode is past the point of diminishing returns.

The chip IPC shown in Figure 3 below each of the multicore designs shows that the 8Kbyte cache is indeed the optimal design because it maximizes the chip IPC.

Although our example used a single benchmark for illustration, in general, a design can use average results from a benchmark suite to obtain an optimal design for the entire benchmark set.

The example above demonstrates that multicore designs must carefully allocate chip area between a core's resources. The existing single processor design approach of building ever increasing caches (and constructs such as extremely deep pipelines, many-ported register files, etc.) can be counter productive to power efficiency and performance. In fact, because the multicore choice did not exist, traditional sequential processors have ventured far beyond the point of diminishing returns for many single core mechanisms such as caches, issue width, and pipeline depth. As an example, caches in many commercial sequential processors occupy over two-thirds of the die area. The kill rule suggests that future core designs for multicore will use significantly simpler cores with significantly smaller caches.

Conversely, this example also demonstrates that simply increasing the number of cores without creating the right balance of resources within a core (and hence the core size) can be a wasteful exercise. There is a danger that the number of cores will become the new MHz paradigm – the Kill rule demonstrates that the number of cores can become yet another meaningless indicator of performance.

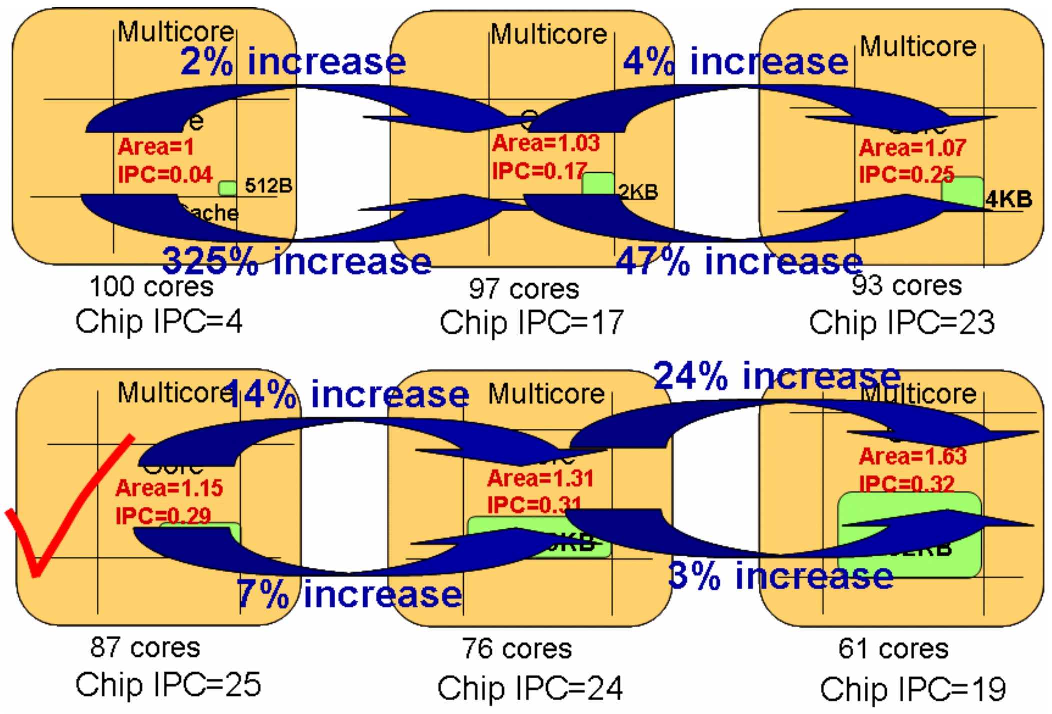

Figure 3 The Kill Rule suggests that the multicore with 8Kbyte caches per core is the optimal design point for MPEG2 decode since increasing the cache further does not yield a proportional increase in core performance

3. SUMMARY

In the days of single core designs, architects would increase core resource sizes (e.g., cache size) as more transistors became available in each succeeding technology generation. Multicores provide the further choice of including additional cores. Thus, given more area, increasing the number of cores and keeping resource sizes relatively small might result in more performance than increasing resource sizes and keeping the number of cores constant. This new dimension for improving performance and power efficiency in multicore requires us to rethink processor architecture and is captured in a simple principle called the Kill Rule. The Kill Rule states that a resource size must be increased only if for every 1% increase in core area there is at least a 1% increase in core performance.

The kill rule suggests that future core designs for multicore will use significantly simpler cores with significantly smaller caches. Processors in the early 90's sported 5 to 6 pipeline stages, and 16 to 32Kbyte caches. Compare that to todays 30+ pipeline stages and multi Megabyte caches. Our best guess is that the optimal design point for the future will reflect processor designs from the mid 90's.