

# Co-scaling Threads and Frequency to Tame Power Consumption

Ramakrishna Kotla    Ravi Kokku    Mike Dahlin  
The University of Texas at Austin

High-throughput applications (like web servers, database servers, and routers) often exhibit two characteristics. First, the workloads processed by these applications fluctuate significantly; the peak processing requirements are often substantially higher than the average. To facilitate running the processors at low power levels during intervals of low load, most modern systems hosting such applications employ processors with frequency and voltage scaling mechanisms. Second, applications exhibit a high degree of parallelism. For instance, requests on different connections to a web server can be processed in parallel. To exploit parallelism and achieve high throughput, systems hosting such applications are increasingly utilizing processors that are designed using *simultaneous multi-threading (SMT)*; SMT-based systems contain multiple independent threads per processor that can execute in parallel.

For such systems, this poster identifies the right technique of combining frequency scaling and thread scaling to minimize power consumption, while achieving the desired throughput. We derive this technique in the following paragraphs.

**Achieving the desired throughput:** From a given level of thread allocation and frequency setting, a system can increase the throughput of the applications by scaling-up either (1) frequency and voltage of the processor, or (2) the number of threads that serve the applications. We make two observations. First, for a given number of threads, throughput increases linearly with frequency (and voltage) as long as the workload is compute-bound, and flattens when the I/O overheads dominate (i.e., workload becomes I/O-bound). Second, at a given frequency, throughput increases linearly with threads as long as the workload is I/O-bound, and saturates when either the processor or a shared resource becomes the bottleneck. Hence, *frequency/voltage scaling and thread scaling are complementary and should be combined for maximum benefit.*

**Minimizing power:** We first study the effect of the above scaling techniques on power consumption. Consider the following power model that represents all CMOS-based processor designs.

$$P(f, \tau) = P_s + C \cdot V_{dd}^2 \cdot f \cdot U(\tau) \quad (1)$$

where  $P_s$  is the static power,  $C$  is the capacitance,  $V_{dd}$  is the supply voltage,  $f$  is the clock frequency,  $U$  is the utilization of processor functional units. In SMT-based processors,  $U$  increases linearly with the number of activated threads

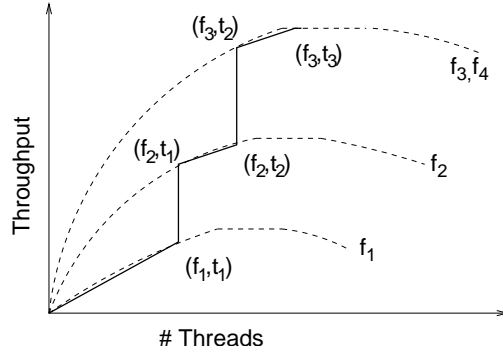


Figure 1: Dotted lines show the variation of throughput with threads and frequency. Solid line shows the right behavior of an adaptive system that achieves the desired throughput while minimizing the power consumption.

$\tau$ ; multiple threads make better utilization of the processor functional units. Further, for CMOS circuits,  $V_{dd}$  is proportional to  $f$ . Hence, Equation 1 can be rewritten as

$$P(f, \tau) = P_s + K \cdot f^3 \cdot (L_0 + L_1 \cdot \tau) \quad (2)$$

where  $K$  is a constant, and  $L_0$  and  $L_1$  represent the contribution of thread-independent and thread-dependent parts of the processor to the power consumption.

From Equation 2, the power consumed by a system varies cubically with frequency and linearly with threads. Hence, *to increase the throughput, it is more power-efficient to scale-up threads than frequency.*

**Co-scaling threads and frequency:** The above two conclusions define in two steps the right way of scaling-up threads and frequency—(1) at a given frequency, increase the threads till further increase does not significantly improve the throughput, and (2) increase the frequency, and repeat step 1. The solid line in Figure 1 depicts the scaling-up behavior. To scale-down threads and frequency, the system follows exactly the reverse path—it reduces threads and frequency in turn to reach the right frequency and thread allocation to meet the current requirements. We will present more details about the technique in the poster.

Realizing such a technique requires mechanisms to measure the throughput and power of the system, and policies to determine *how frequently* to check for thread allocation, *how many* threads to allocate/release at a given time, *when* to adjust the frequency level, and *what* should the new frequency setting be? We will address some of these questions in the poster.