# Scaling and Packing on a Chip Multiprocessor*

Vincent W. Freeh† Tyler K. Bletsch† Freeman L. Rawson, III‡

†Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206
{vwfreeh,tkbletsc}@ncsu.edu

‡IBM Austin Research Laboratory
11501 Burnet Road
Austin, Texas 78758
frawson@us.ibm.com

## Abstract

*Power management is critical in server and high-performance computing environments as well as in mobile computing. Many mechanisms have been developed over recent years to support a wide a variety of power management techniques. In particular, general purpose microprocessors now support dynamically modifying the power-performance state through voltage and frequency changes. This development spawned a very important area of this research in* dynamic voltage and frequency scaling *(DVFS). On the other hand, in a multiprocessor environment one can perform power management by offlining and idling processors when computational demand is low in a technique called* CPU packing. *This paper examines the effect of combining voltage and frequency scaling and CPU packing in a multiprocessor. Furthermore, it examines DVFS on a* chip multiprocessor *in which multiple processor cores are placed on a single die. This paper shows that in general one should use DVFS first, then CPU packing. Furthermore, we find that the effectiveness of CPU packing is application-dependent: commercial workloads (e.g. Apache) with periods of low utilization can reduce power by as much as 19% via packing, while the improvement to HPC workloads ranges from small to negligible.*

## 1 Introduction

Power-aware computing is important in both demand-driven commercial data centers and throughput-oriented high-performance computing centers because of positive effects on cost, maintainability, and reliability. As less energy is used, the reduction in the cost of electricity is not the only benefit. Less energy consumed means less heat generated (and less energy spent cooling the equipment). Less heat leads to greater mean time between failures for components as well as the possibilities for greater component density.

This is a practical problem. Even relatively new centers that house large clusters are having trouble staying within power and cooling constraints [39]. Historically, data centers have tried to push performance at all costs. Unfortunately, the "last drop" of performance tends to be the most expensive. One reason is the cost of power consumption, because power is proportional to the product of the frequency and the square of the voltage. As an example of the problem that is faced, several years ago it was observed that on their current trend, the power density of a microprocessor will reach that of a nuclear reactor by the year 2010 [20].

To balance the concerns of power and performance, new architectures have aggressive power controls. One common mechanism on newer microprocessors is the ability of the application or operating system to select the frequency and voltage on the fly. This technique is called *dynamic voltage and frequency scaling* (DVFS). We denote each possible combination of voltage and frequency a performance state, or *p-state*.

While changing p-states has broad utility, including extending battery life in small devices, the primary benefit of DVFS for cluster computing occurs when the p-state is reduced in regions where the CPU is not on the critical path. In such a case, power consumption will be reduced with little or no reduction in end-user performance. Previously,

*This research was supported in part by IBM UPP and SUR awards.

p-state reduction has been studied in code regions where the bottleneck is in the memory system [24, 22, 7, 17, 16], between nodes with different workloads [28], or during communication phases [33].

Newer clusters are often built with multiprocessor nodes. With the introduction of *chip multiprocessors* (CMP), where several CPU cores are placed on a single die, multiprocessor clusters will be ubiquitous in the near future. However, power management via DVFS on multiprocessors is not well understood.

Orthogonally to this, a multiprocessor can manage power using *CPU packing* [18]. In this technique tasks are packed onto some active processors, while inactive processors are idled or powered down. Thus, packing provides a finite number of power-performances levels, which we call *configurations*.

Packing is quite similar to scaling. Both techniques provide a finite number of discrete reduced power states, but reducing power generally has a negative effect on performance. Mechanisms developed for frequency-voltage scaling are easily adapted to CPU packing. However, as stated above, a reduced p-state is more power efficient, but a reduced configuration is generally *less* power efficient. In packing, the total power to the processors is greatly reduced, but the power used by other components (such as the power supply) is only slightly reduced. When this supporting power is considered, the net power consumption per processor (or, equivalently, the power-performance ratio) increases as one packs. While packing makes the machine less efficient, the machine consumes less power. Thus, when the demand for performance is low enough that a packed configuration is sufficient, packing is desirable.

This paper examines the interplay of scaling and packing. This paper also examines power management for a CMP. The results shown in this paper are real-world power measurements of a physical cluster. It shows that it is more efficient to scale first and then pack. In throughput computing where one wants to execute a set of tasks as efficiently as possible, scaling is often sufficient. However, in demand-driven commercial applications, there is not a set of tasks. Rather, there is only a current demand. When demand is low, there is a distinct and significant advantage to packing. Our system shows a savings of 19% due to packing when the demand is one-fourth of the maximum of the server's total capacity.

The rest of this paper is organized as follows. The next section discusses relevant related work. Section 3 explains our testing methodology, with the results of these tests following in Section 4. The last section offers some conclusions.

## 2 Related Work

Most prior work focuses either on HPC applications and installations or on commercial ones. This paper looks at both because while the policies are different, the mechanisms and machines are the similar. A commercial installation tries to reduce cost while servicing client requests. On the other hand, an HPC installation exists to speed up an application, which is often highly regular and predictable.

Several researchers have investigated saving energy in server-class systems or computing clusters. The basic idea is that if there is a large enough cluster of such machines, such as in hosting centers, energy can become an issue. In [8], Chase *et al.*, among other things, determine the aggregate system load and then determine the minimal set of servers that can handle that load. All other servers are transitioned to a low-energy state. A similar idea leverages work in cluster load balancing to determine when to turn machines on or off to handle a given load [37, 38]. Elnozahy *et al.* [12] investigated the policy in [37] as well as several others in a server farm. They found that allowing each node to independently set its voltage performed almost as well as more complicated schemes that required coordination between server nodes. Such work shows that power and energy management are critical for commercial workloads, especially web servers [3, 32]. Additional approaches have been taken to include DVS [11, 40] and request batching [11]. The work in [40] is interesting because it applies real-time techniques to web servers in order to conserve energy while maintaining quality of service. We have shown that throttling the CPU can increase performance given a power limit [14].

One approach is to save energy in an application-specific way; Chen et al. used this approach for a parallel sparse matrix application [9]. Another HPC effort that addresses the memory bottleneck is given in [23]; however, this is a purely static approach. This paper also introduced the concept of *CPU-criticality*, which was adapted for HPC in [24]. Also, Cameron et al. [15] developed a measurement infrastructure for power-aware HPC programs on a cluster of laptops and performed experiments on NAS programs. In [26], a generic evaluation infrastructure that makes use of SimPoint [41] is described; this infrastructure combines simulation and direct measurement. Another approach that can be used for HPC and is independent of the application can be found in [30]; performance counters are used to estimate IPC dynamically and choose the correct gear. Finally, metrics for power-aware HPC are described in [25].

There are numerous studies of energy and power consumption [2, 27, 34, 35, 36, 43]. Many researchers reduce power with novel microarchitectures [29, 6, 13, 21]. A compiler can generate power-aware code [19, 44, 45]. There

are several general power/energy management systems and tools [1, 4, 5, 10].

Cameron et al. [7] uses a variety of different DVFS scheduling strategies (for example, both with and without application-specific knowledge) to save energy without significantly increasing execution time. A similar run-time effort is due to Hsu and Feng [22]. Our own prior work is fourfold: an evaluation-based study that focused on exploring the energy/time tradeoff in the NAS suite [17], development of an algorithm for switching p-states dynamically between phases [16], leveraging load imbalance to save energy [28], and minimizing execution time subject to a cluster energy constraint [42].

Every generation the microarchitecture feature size decreases but the chips area remains roughly constant. Thus microarchitects must choose what to do with additional transistors. Further increasing the clock frequency requires ever more complex designs and has diminishing returns. As a result the current trend embraces parallelism. The *chip multiprocessor* (CMP) [31, 47] is naturally the first such effort because the most straightforward way to chip-level parallelism. A CMP is merely the co-location of more than one entire CPU core (including caches) on a single chip.

## 3 Methodology

The testbed for this paper is a 16-node Opteron cluster connected by gigabit ethernet. Each node has two dual-core Opteron processors (model 265) for a total of 4 CPUs. Each core has private L1 (128 KB) and L2 (1 MB) caches. The Opteron supports DVFS scaling in 5 p-states having frequencies ranging from 1.8 to 1.0 GHz. The design of the chip is such that the frequency is changed per socket, not per core. Thus, both cores on a single dual-core processor are always operating in the same p-state. This paper uses the term *core* to denote a single CPU or microprocessor. The term *socket* is used to denote the physical medium that contains cores. A *node* is a complete machine consisting of sockets, memory, network interface card, etc. Finally, a *cluster* is a collection of nodes. Thus the testbed is a cluster of 16 nodes, each node has two sockets, each socket has two cores. There are a total of 64 CPU cores in this cluster.

Figure 1 shows a schematic layout of each node. Half of the memory is directly connected to each socket, which the socket can access through its memory bus. To access the other half, it is necessary to use the HyperTransport connection between sockets. Thus in this system there is 1GB of *local* memory and 1GB of *remote* memory. Including the caches, the memory hierarchy consists of four levels (L1, L2, local, and remote).

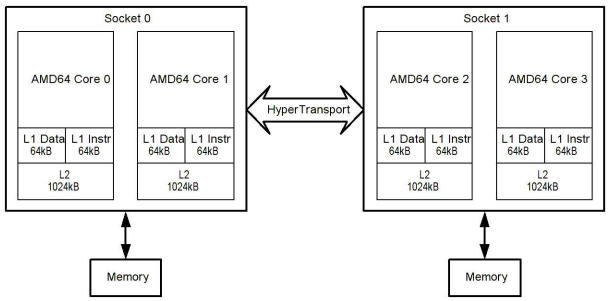



**Figure 1. Opteron microarchitecture.**

The power consumed by the cluster is measured using WattsUp meters. The meters provide online power usage data through a serial connection. Thus, the testbed collects true power consumption data in real-time. Software integrates power usage over time to determine energy consumption.

Our methodology uses several benchmarks in three classes of tests. The first class consists of single-threaded, computation-bound programs built around simple algorithms. In this class are some basic linear algebra programs, notably DAXPY, and some comparable hand-built programs. The purpose of this class is to establish baseline performance. In these programs the CPU is almost always on the critical path; therefore, scaling and packing (which reduce CPU performance) should have the maximal detrimental effect on application performance.

The second benchmark class consists of parallel high-performance benchmarks from the NAS parallel benchmark suite [46]. The NAS suite is a popular high-performance computing benchmark. It consists of scientific benchmarks including application areas such as sorting, spectral transforms, and fluid dynamics. The purpose of this class is to evaluate scaling and packing in an HPC environment.

The final class consists of a web server, namely Apache. The web server is loaded with the *mod_php* module to support dynamic pages constructed in PHP script—a typical installation. In addition to the web server there is a client program that make requests at a parameterizable rate. The purpose of this class is to evaluate scaling and packing in a commercial data center.

**Node states** There are two dimensions to the reduced power-performance states for a node: one due to scaling the other from packing. There are 5 scaling p-states denoted by frequency: every 200MHz from 1.8 to 1.0 GHz. It is

implicitly understood that the voltage is reduced along with the frequency.

There are five additional packing states, called *configurations*. The first configuration has 4 *active* cores. It is denoted as $\times$**4**—read as "by 4" cores. In the next configuration, $\times$**3**, only 3 cores are active. Because all cores are equivalent, it does not matter which core is idled, so there is only one 3-core configuration. Similarly, $\times$**1** denotes all configuration in which only 1 core is active. However, there are two distinct ways to configure two active cores. With mask $\times$**2** the two active cores are on the same socket, whereas with $\times$**2*** one core on each socket is active and one is idle. (This configuration uses both sockets, more resources, so it has a longer denotation than $\times$**2**.) Configuration $\times$**2*** has 1GB of local memory for each core, while in $\times$**2**, the two cores have to share the 1GB of local memory. The $\times$**2** configurtion tends uses less power, because one socket is idle (and possibly even powered down).

When multiple nodes are used, the number of nodes will be prepended to the packing configuration. For example, "4$\times$**2**" means "4 nodes, 2 cores per node" for a total of 8 cores.

Due to the limitations of this particular Opteron processor, the frequency (p-state) of cores on the same socket must be the same. Therefore, we only need to consider two frequency pairs. In three packing configurations both sockets are active, thus there are 25 power-performance states or *node states* for each configuration. In the other two configurations, there are 5 unique p-states. As a result, there are 85 possible node states. However, non-homogeneous scaling (sockets use different p-states) is beneficial only when load is severely unbalanced. Restricting the system to homogeneous scaling means, there are only 25 node states. Furthermore, our results show that most of these remaining node states are not effective—that is there is another node state that provides superior performance while using less power.

# 4 Results

We find that the effectiveness of DVFS and CPU packing depends greatly on the type of workloads (CPU-bound, HPC, or commercial). The results for each of these classes are discussed in turn.

## 4.1 CPU-bound workloads

Figure 2 presents results for a computation-bound application. It is a DAXPY program, which is a small kernel from linear algebra. In this program the CPU is the critical resource. There is no I/O and the memory footprint fits in cache. Therefore, a reduction in the raw processing performance (by either scaling or packing) will have the maximal detrimental effect on performance. Only one program is shown; however, the results are representative of the whole class of CPU-bound programs.

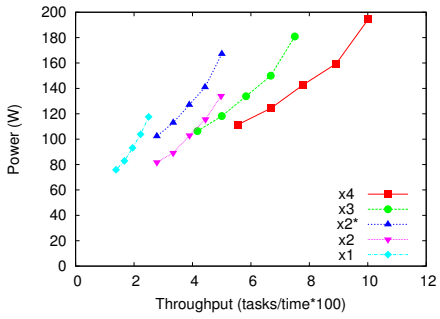



**Figure 2. Computation-intensive program (daxpy).**

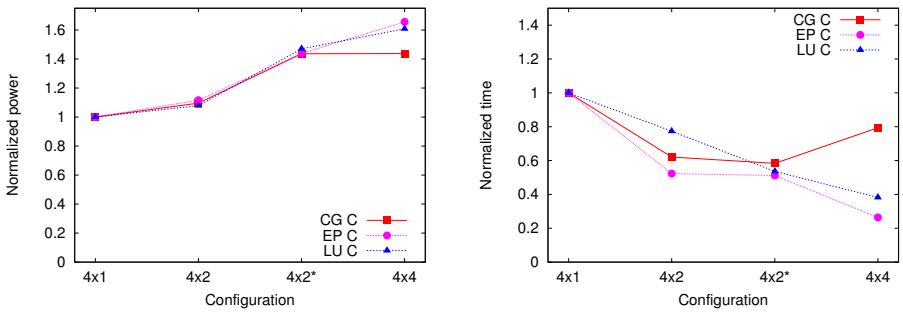
The figure shows a scatter plot of power consumption versus throughput for the 25 homogeneously scaled node states. It reports on five packing configurations, with five p-states for each configuration. The five points for each configuration are joined by a line. The highest point (which uses the most power) is 1.8GHz and the lowest point is 1.0GHz.

The figure shows the average power consumption over the entire test, but there is very little difference between the maximum and the minimum. Throughput is shown in tasks (test programs) completed per second (which is arbitrarily scaled by 100). There is one program run on each active processor and the programs take approximately 40 seconds to execute at the top frequency (1.8GHz).

The best node state for a given power level is found by scribing a horizontal line at the power level and selecting the rightmost node state (that with the greatest throughput for a given power consumption).[1] Conversely, the best node state for a particular throughput demand is the lowest configuration (consumes the least power). Figure 2 clearly shows that as throughput demand decreases, one should *scale-first then pack*. It also shows that 3-active cores ($\times$**3**) is not an effective configuration.

In this test there is almost no demand on the memory (beyond L2). Therefore, the 2-core configurations achieve similar throughput levels. However, when both active cores are on the socket, the other socket can be idled or powered

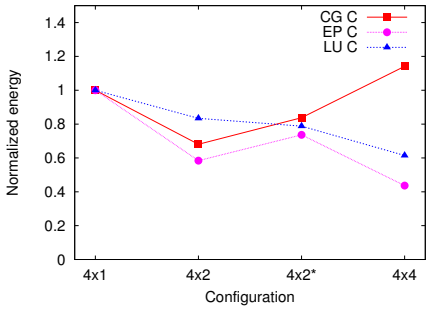
[1] This explanation is correct if the lines on the graph represented a continuum, which they do not. However, the correct description for discrete points is complex, uninteresting, and unnecessary.



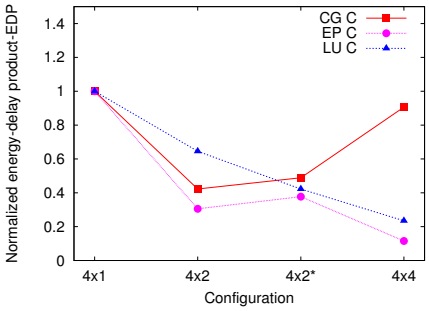

(a) Power (base CG=592W, EP=580W, LU=595W)




(b) Time (base CG=278s, EP=170s, LU=977s)




(c) Energy (base CG=165KJ, EP=99KJ, LU=581KJ)




(d) ED product (base CG=46MJs, EP=17MJs, LU=567MJs)

**Figure 3. Packing of HPC workloads on a single node. Normalized results for 3 NPB benchmarks at 1.8GHz.**

down. This shows the powered-down case, which saves at least 20W. Later, this paper shows results where the performance of $\times$**2*** is superior to $\times$**2** due to the demand on the memory subsystem.

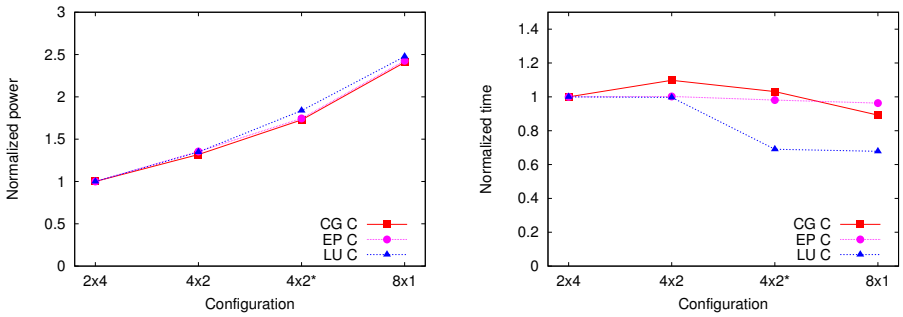
## 4.2 HPC workloads

The results for the two one-socket configurations above were obtained by physically removing the unused chip. Our present environment does not allow us to power a CPU completely off, but there is no technical reason that prevents this. In order to investigate the true potential of CPU packing with the idle socket powered down, the results in subsequent figures "simulate" powering down the other socket by subtracting 20W from the actual measured power.

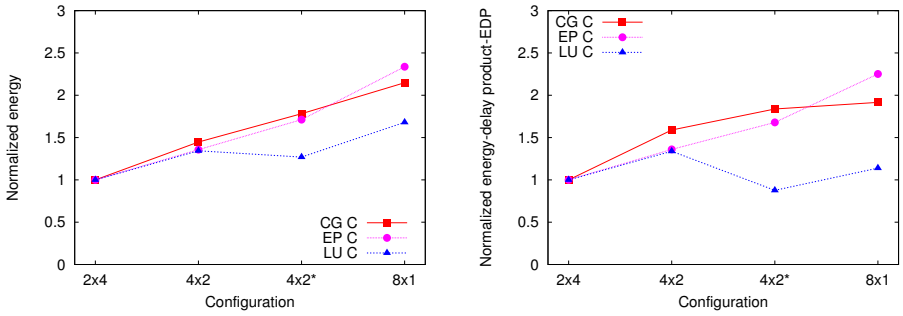
Figure 3 shows the effect of packing on throughput computing or batch-oriented tasks as representative of those in a high-performance computing center. There are four subgraphs showing average power, elapsed time, energy consumed, and energy-delay product (EDP). Only time and energy are measured, average power is the quotient of energy over time and EDP is the product of energy and time. The x-axis is the configuration, ordered in increasing amount of resources used. It actually shows the number of nodes used (4 in this case) and the packing configuration. Thus the left-most configuration uses four cores, one on each of four nodes. The right-most uses 16 cores, four on each of four nodes.

Each plot shows the value relative to the smallest configuration on the y-axis. There are data for three of the NAS programs, selected because each produces a distinct effect. The absolute values of the normalized points are shown in the captions. These results are for 1.8Ghz, but the results for other frequencies are very similar.

As expected, the power consumption increases as additional resources are used. Unsurprisingly, the time decreases with the increase in resources—except for an anomalous point for CG. Only LU shows a distinct difference between $\times$**2** and $\times$**2***. The latter configuration is much faster because LU has a large demand on memory bandwidth. In $\times$**2** both processes compete for the same path to memory, whereas in $\times$**2*** each process has its own local channel. This is not a property of the LU program. Rather, is a function of the working set size. When this same sized LU problem is




(a) Power (base CG=426W, EP=412W LU=412W) (b) Time (base CG=157s, EP=89s, LU=758s)




(c) Energy (base CG=67KJ, EP=37KJ, LU=312KJ) (d) ED product (base CG=11MJs, EP=3MJs, LU=236MJs)

**Figure 4. Comparing configurations using same number of processors (8). Normalized results for 3 NPB benchmarks at 1.8GHz.**

decomposed onto more nodes, the per-process working set decreases to a point where there is little difference between the times for the two 2-core configurations.

These graphs show that there is little or no benefit to packing onto one core because packing reduces the performance more than it reduces the power consumption. Whether to pack onto two cores appears to be something that must be judged on a case by case basis.

In the previous analysis, packing uses fewer total cores. So, for example, it compares a 4-core, 4-process instance to a 16-core, 16-process instance. Figure 4 fixes the number of cores and processes at 8. The same four packing configurations are shown, but now the number of nodes (the number before the $\times$) varies. We show the same four metrics as above. The x-axis is again sorted in order of increasing resources, but the order in not the same as above. As expected, the power increases as the resources increase. The time does not change much at all for EP because it is almost perfectly CPU-bound. (Note that the graphs in figure 3 (where cores and processes are fixed at 4) are nearly identical.)

CG is interesting because of the trade-off that exists as resources are added. CG gets worse because there is an increase in communication as the number of processes increases. However, as the number of processes increases so do the resources (notably memory), so the working set per process decreases, making the memory hierarchy more efficient. This efficiency eventually offsets the increased communication cost. LU again shows a tremendous improvement in execution time for the two cases where the process does not have to share the socket's memory bandwidth. Nevertheless, the most efficient configuration is $\times$**4**, or using all available resources on each node.

### 4.3 Commercial workloads

The results above do not make a strong case for packing in throughput computing. However, the commercial data center is driven by highly fluctuating, unpredictable demand. Figure 5 presents data from a web server (Apache). The x-axis shows throughput and the y-axis power consumption. For a given node state (configuration and frequency), the demand is varied from zero to maximum and the power is measured.

The lines in this graph connect different points than the lines in Figure 2. Previously, there was only one point for each node state (configuration, p-state pair) because the demand on the system is implicit in the application. Hence, the lines in Figure 2 connect all 5 p-states in the same configuration. In Figure 5, however, there are many points for each node state corresponding to the varying demand, so the lines connect all points for the same node state. Each node configuration line is denoted by the number of active cores and the CPU frequency (so "2 @ 1.6GHz" means two cores, each operating at 1.6GHz).

In this test we only examine two configurations: 4-cores (×**4**) and two-cores on one socket (×**2**). The other two-core configuration (×**2***) is inferior in this test, so it was not shown. However, we do not claim that as a general rule. Our system activates and de-activates cores from the system by modifying the CPU affinity of the apache processes. The 2-core results simulate powering down the unused processor by subtracting a fixed 20W.

First, note that for all node states, power consumption is a nearly linear function of the demand (and resultant throughput). For the top line (×**4** at 1.8GHz) there is a 40W difference. This graph also shows the inefficiency of packing. The maximum performance is 345 replies per second at 187W. The maximum performance with two core has a rate of 196 $s^{-1}$ at 131W, or 57% of the performance and 70% of the power. Four cores at 1.0GHz delivers more performance (210 $s^{-1}$, 61%) for less power (128W, 68%). However, we see that for low demand there is a distinct advantage to packing. For example, at a demand of 90 $s^{-1}$, packing saves 21W or 19%.

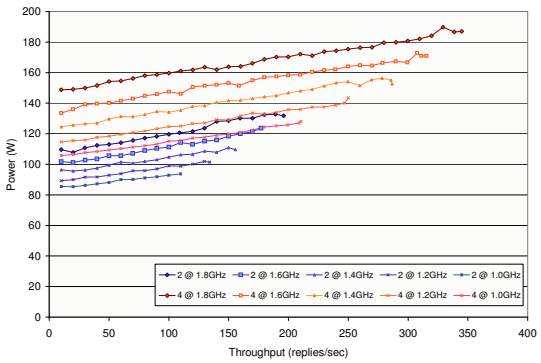



**Figure 5. Apache throughtput test.**

## 5 Conclusion

This paper examines the combination of dynamic frequency and voltage scaling (DVFS) and CPU packing in a multiprocessor environment. It shows that packing is less efficient that scaling, meaning that packing is only practical when utilization is low. While this may or may not be the case for HPC applications, packing is quite effective and arguably necessary in dynamic, demand-driven environments such as a web server.

In short, we show that one should *scale first, then pack* and that *packing is useful only when demand is low*.

## References


[1] F. Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the 9th ACM SIGOPS European Workshop*, September 2000.

[2] L. Benini, A. Bogliolo, and G. De Micheli. Monitoring system activity of OS-directed dynamic power management. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISPLED '98*, 1998.

[3] Pat Bohrer, Elmootazbellah Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, Chandler McDowell, and Ram Rajamony. *Power Aware Computing*, chapter The Case of Power Management in Web Servers. Kluwer/Plenum, 2002.

[4] D.J. Bradley, R.E. Harper, and S.W. Hunter. Workload-based power management for parallel computer systems. *IBM Journal of Research and Development*, 47(5):703–718, September 2003.

[5] D. Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of 27th International Symposium on Computer Architecture(ISCA)*, pages 209–220, 2000.

[6] T. D. Burd and R. W. Brodersen. Energy efficient CMOS microprocessor design. In *Proceedings of the 28th Annual Hawaii International Conference on System Sciences*, pages 288–297. IEEE, IEEE Computer Society Press, January 1995.

[7] K.W. Cameron, X. Feng, and R. Ge. Performance-constrained, distributed dvs scheduling for scientific applications on power-aware clusters. In *Supercomputing*, November 2005.

[8] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. In *Symposium on Operating Systems Principles*, pages 103–116, 2001.

[9] Guilin Chen, Konrad Malkowski, Mahmut Kandemir, and Padma Raghavan. Reducing power with performance contraints for parallel sparse applications. In *First Workshop on High-Performance, Power-Aware Computing*, April 2005.

[10] Rita Yu Chen, Mary Jane Irwin, and Raminder S. Bajwa. Architecture-level power estimation and design experiments. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 6:50–66, January 2001.

[11] Elmootazbellah Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy conservation policies for web servers. In *USITS '03*, 2003.

[12] E.N. (Mootaz) Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy-efficient server clusters. In *Workshop on Mobile Computing Systems and Applications*, February 2002.

[13] G. Semeraro et. al. Energy efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *Proceedings of 8th International Symposium on High Performance Computer Architecture (HPCA)*, pages 29–40, February 2002.

[14] Mark E. Femal and Vincent W. Freeh. Safe overprovisioning: Using power limits to increase aggregate throughput. In *Workshop on Power-Aware Computer Systems*, December 2004.

[15] X. Feng, R. Ge, and K. W. Cameron. Power and energy of scientific applications on distributed systems. In *International Parallel and Distributed Processing Symposium*, April 2005.

[16] Vincent W. Freeh, David K. Lowenthal, Feng Pan, and Nandani Kappiah. Using multiple energy gears in MPI programs on a power-scalable cluster. In *Principles and Practices of Parallel Processing (*PPOPP, Chicago, IL, June 2005.

[17] Vincent W. Freeh, David K. Lowenthal, Rob Springer, Feng Pan, and Nandani Kappiah. Exploring the energy-time tradeoff in MPI programs on a power-scalable cluster. In *IPDPS*, Denver, CO, April 2005.

[18] Soraya Ghiasi and Wes Felter. Cpu packing for multiprocessor power reduction. In *PACS 2003 (Power-Aware Computer Systems)*, pages 117–131, San Diego, CA, December 2003.

[19] Chris Gniady, Y. Charlie Hu, and Yung-Hsiang Lu. Program counter based techniques for dynamic power management. In *Proceedings of the 10th International Symposium on High-Performance Computer Architecture*, February 2004.

[20] Richard Goering. Current physical design tools come up short. *EE Times*, April 14 2000.

[21] Michael K. Gowan, Larry L. Biro, and Daniel B. Jackson. Power considerations in the design of the alpha 21264 microprocessor. In *Proceedings of 35th Design Automation Coference (DAC)*, pages 726–731, June 1998.

[22] Chung hsing Hsu and Wu chun Feng. A power-aware run-time system for high-performance computing. In *Supercomputing*, November 2005.

[23] C-H. Hsu and U. Kremer. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. In *ACM SIGPLAN Conference on Programming Languages, Design, and Implementation*, June 2003.

[24] Chung-Hsing Hsu and Wu-chun Feng. Effective dynamic-voltage scaling through CPU-boundedness detection. In *Fourth IEEE/ACM Workshop on Power-Aware Computing Systems*, December 2004.

[25] Chung-Hsing Hsu and Wu-chun Feng. Towards efficient supercomputing: Choosing the right efficiency metric. In *First Workshop on High-Performance, Power-Aware Computing*, April 2005.

[26] Chunling Hu, Daniel Jimenez, and Ulrich Kremer. Toward an evaluation infrastructure for power and energy optimizations. In *First Workshop on High-Performance, Power-Aware Computing*, April 2005.

[27] R. Joseph and M. Martonosi. Run-time power estimation in high performance microprocessors. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISPLED '01*, August 2001.

[28] Nandani Kappiah, Vincent W. Freeh, and David K. Lowenthal. Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs. In *Supercomputing 2005 (to appear)*, November 2005.

[29] Peter M. Kogge, Kanad Ghose, and Vincent W. Freeh. Morph: Adding an energy gear to a high-performance microarchitecture for embedded applications. In *Kool Chips Workshop, MICRO-33*, Monterey, CA, December 2000.

[30] Ramakrishna Kotla, Soraya Ghiasi, Tom Keller, and Freeman Rawson. Scheduling processor voltage and frequency in server and cluster systems. In *Workshop on High-Performance, Power-Aware Computing (HPPAC)*, 2005.

[31] David Lammers. Ibm to unveil power4 processor at hot chips. EE Times Online, August 1999. http://www.eetimes.com/story/OEG19990804S0023.

[32] Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kistler, and Tom W. Keller. Energy management for commerical servers. *IEEE Computer*, pages 39–48, December 2003.

[33] Min-Yeol Lim, Vincent W. Freeh, and David K. Lowenthal. Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs. In *Supercomputing (SC—06)*, Tampa, FL, November 2006.

[34] J. Lorch. A complete picture of the energy consumption of a portable computer. Master's thesis, University of California at Berkeley, 1995.

[35] Akihiko Miyoshi, Charles Lefurgy, Eric Van Hensbergen, Ram Rajamony, and Raj Rajkumar. Critical power slope: Understanding the runtime effects of frequency scaling. In *Proceedings of the 16th International Conference on Supercomputing*, pages 35–44, 2002.

[36] Trevor Mudge. Power: A first class architectural design constraint. *IEEE Computer*, 34(4):52–57, April 2001.

[37] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Dynamic cluster reconfiguration for power and performance. In *Compilers and Operating Systems for Low Power*, September 2001.

[38] Eduardo Pinheiro, Ricardo Bianchini, Enrique V. Carrera, and Taliver Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Workshop on Compilers and Operating Systems for Low Power*, September 2001.

[39] Thermal management and server density: Critical issues for today's data center. http://www.rackable.com/Rackable_WPaper.pdf, 2004.

[40] Vivek Sharma, Arun Thomas, Tarek Abdelzaher, and Kevin Skadron. Power-aware QoS management in web servers. In *24th Annual IEEE Real-Time Systems Symposium*, Cancun, Mexico, December 2003.

[41] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 45–57, 2002.

[42] Robert C. Springer IV, David K. Lowenthal, Barry Rountree, and Vincent W. Freeh. Minimizing execution time in MPI programs on an energy-constrained, power-scalable cluster. In *ACM Symposium on Principles and Practice of Parallel Programming*, March 2006.

[43] P. Stanley-Marbell and M. S. Hsiao. Fast, flexible, cycle-accurate energy estimation. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISPLED '01*, August 2001.

[44] C. Su, C. Tsui, and A. Despain. Low power architecture and compilation techniques for high-performance processors. In *Proceedings of the IEEE COMPCON*, February 1994.

[45] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: a first step towards software power minimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4):437–445, 1994.

[46] F. C. Wong, R. P. Martin, R. H. Arpaci-Dusseau, and D. E. Culler. Architectural requirements and scalability of the NAS parallel benchmarks. In *Proceedings of Supercomputing '99*, Portland, OR, November 1999.

[47] Mike Zazaian. Intel: 80 cores by 2011. TechFreep Online, September 2006. http://techfreep.com/intel-80-cores-by-2011.htm.