Providing QoS with Virtual Private Machines

Kyle J. Nesbit, James Laudon*, and James E. Smith

University of Wisconsin – Madison
Dept. Electrical and Computer Engineering
{ nesbit, jes }@ece.wisc.edu

*Sun Microsystem**
james.laudon@sun.com

1. Motivation

To provide both efficiency and high throughput, CMP-based systems exploit resource sharing, especially in the memory hierarchy. Shared resources include both bandwidth, including interconnection paths with associated buffering, and storage space. Resource sharing leads to interference among threads so that a thread's performance depends on the threads with which it is co-scheduled. Although main memory space is managed by operating system algorithms, most of the other memory resources – main memory bandwidth and all cache spaces and bandwidths – are typically managed through hardware mechanisms.

The weakening of software control over shared hardware resources comes at a time when trends suggest predictable performance and equitable resource management will be critical for evolving and emerging applications. To illustrate this, consider the following applications and application trends.

1) Applications with soft real-time constraints, for example those that support multimedia such as HD videos and resource-intensive video games, will become more important in the future as consumer computing moves away from the desktops and towards game consoles (digital hubs) and portable devices (cell-phones). It is important that a system based on multi-threaded chips can provide assured performance levels for certain threads regardless of what other threads are doing.

2) Fine-grain parallel applications are the key to the long-term success of increasingly multi-threaded chips at the client level. To extract fine-grain parallelism from an application, it is important that developers can rely on predictable execution times in order to effectively schedule concurrent tasks and optimize synchronization overheads.

3) The availability of inexpensive multiprocessor systems will create a new wave of server consolidation and hosted applications. In these workloads, a server supports multiple threads on behalf of independent customers. It is important that resources be shared in a controlled and equitable manner so that customer tasks perform in a responsive, timely way, and that customers receive the service they pay for, regardless of what other customers are doing.

Consequently, future CMP-based systems must incorporate hardware mechanisms and software policies to provide threads with *Quality of Service (QoS)*. QoS is necessary to preserve performance predictability and facilitate the design of dependable systems.

2. Virtual Private Machines

To provide a solution, we propose a QoS framework based on Virtual Private Machines (VPMs). A VPM is defined as a set of allocated resources (notably processors, bandwidths, and memory spaces). The key objective is that a VPM should provide performance at least as good as a real private machine having the same resources.

System software and applications implement policies that determine VPM configurations, and hardware mechanisms enforce the allocations [1]. For example, Figure 1 illustrates a generic CMP-based system and Figure 2 illustrates the CMP is divided into four VPMs. VPM0 is given a significant fraction (50%) of resources to support a demanding multimedia application, while the other three VPMs are assigned a much lower fraction of resources (10% each). This leaves 20% of the cache memory resources unallocated. Overall, VPMs provide system software with a useful abstraction for maintaining control over shared microarchitecture resources.

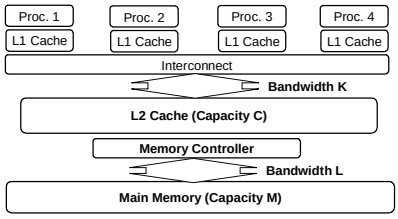

Figure 1: CMP-Based System with Shared Memory System Resources

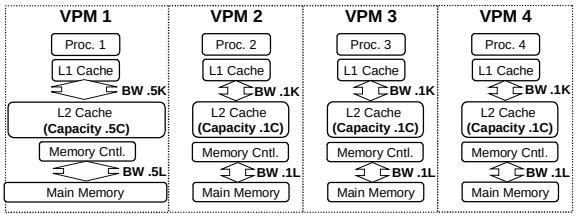

Figure 2: Four Virtual Private Machines

A VPM is not the same as hardware partitioning. In a VPM, a thread is allocated a minimum amount of shared resources, and if excess resources are available, the VPM may receive more than its allocated resources, thus providing additional speedup. Excess resources are resources that are either unallocated, or resources that are allocated to a VPM but are not used. Excess resources are distributed according to an *excess resource policy*.

We briefly describe a few important applications of VPMs, and describe how system software and application developers can control VPMs in order to achieve system-level objectives.

2.1. Soft Real-Time

For single-threaded applications with soft real-time performance requirements, an application developer can use automated tools to determine the minimum VPM resources required to meet the application's performance requirements. When the application is deployed, as long as the application is running inside a validated VPM configuration, the application will meets its performance requirements regardless of the load placed on the CMP by other tasks [2]. An application's VPM configuration is encoded in the application binary, and at run time is part of the application's architected state (via ISA-supported control registers). Whenever the application is context switched in, its VPM configuration is loaded.

VPM hardware mechanisms must be supported by an OS scheduler that takes into account applications' VPM and computation time requirements. Such a scheduling algorithm ensures that the shared CMP resources are never over allocated.

Traditionally, general-purpose OSes use priority levels to ensure that the most critical applications meet their performance objectives. Priority levels do not offer the same control over QoS as VPMs do, but priority levels do provide some information about applications' relative performance requirements. Policies that translate priority levels to VPM configurations is an open research problem.

2.2. Performance Isolation

In server systems, VPMs provide performance isolation. The system software (an OS or VM) allocates individual clients their own VPMs. In this case, a client may be an internet user connected to a streaming server, or a business leasing a fraction of a server to host a scientific or enterprise application. In either case, the client's VPM ensures that the client will receive its allocated (purchased) share of the server regardless of the load placed on the server by other clients.

Performance isolation is also important for other computing environments, such as desktops, as it can be disconcerting to a user when his/her application displays significant performance differences for no obvious reason.

2.3. Excess Resource Policies

As described earlier, excess resource policies control how excess resources are distributed. Excess resources are resources that are not needed to meet a workload's strict performance requirements (e.g. soft real-time requirements).

Excess resource policies depend on system-wide objectives, and in turn, these objectives depend on the system's workload. Two examples of system-wide objectives are to improve *aggregate performance* and *fairness*; sometimes these objectives appear in combination. Because applications have diverse and sometimes disparate resource requirements, hardware should support multiple parameterized excess service policies that can be controlled by system-level software policies.

System software has a global view of resources and applications, which gives software the potential to distribute resources in a globally optimized manner, at least when compared with hardware mechanisms that tend to be based on simple, localized heuristic.

Software policies may partially override the hardware's excess resource policy by allocating resources that would have otherwise been unallocated, although software policies have to rely on the hardware's excess resource policies to distribute excess resources that have been allocated to a thread but are unused. For this reason, VPM software and hardware excess policies should be co-designed.

3. Conclusion

In this paper, we have summarized the VPM framework, including its constituent hardware mechanisms and software policies. It is our position that the VPM framework has the potential to meet the disparate QoS requirements of general-purpose CMP-based systems. Through future research we plan to illustrate the effectiveness and feasibility of the proposed tools, hardware mechanisms, and software policies.

4. References

[1] Kyle J. Nesbit, Nidhi Aggarwal, James Laudon, and James E. Smith, Fair Queuing Memory Systems, In *Proceedings of the 39th Annual International Symposium on Microarchitecture (MICRO-39)*, Dec. 2006.

[2] Jae W. Lee, and Krste Asanovic, METERG: Measurement-Based End-to-End Performance Estimation Technique in QoS-Capable Multiprocessors. *In 12th Real-Time an Embbeded Technology and Applications Symposium* (RTAS-12), April 2006.