A case for coordinated resource management in heterogeneous multicore platforms

Priyanka Tembey, Ada Gavrilovska, Karsten Schwan

A case for coordinated resource management in heterogeneous multicore platforms

Priyanka Tembey
Georgia Institute of Technology

Ada Gavrilovska
Georgia Institute of Technology

Karsten Schwan
Georgia Institute of Technology

Abstract

Recent advances in multi- and many-core architectures include increased hardware-level parallelism (i.e., core counts) and the emergence of platform-level heterogeneity. System software managing these platforms is typically comprised of multiple independent resource managers (e.g., drivers and specialized runtimes) customized for heterogeneous vs. general purpose platform elements. This independence, however, can cause performance degradation for an application that spans diverse cores and resource managers, unless managers coordinate with each other to better service application needs. This paper first presents examples that demonstrate the need for coordination among multiple resource managers on heterogeneous multicore platforms. It then presents useful coordination schemes for a platform coupling an IXP network processor with x86 cores and running web and multimedia applications. Experimental evidence of performance gains achieved through coordinated management motivates a case for standard coordination mechanisms and interfaces for future heterogeneous many-core systems.

1 Introduction

Islands of cores. Recent advances in multi- and many-core architectures include increased hardware-level parallelism (i.e., core counts) and the emergence of platform-level heterogeneity. Examples include the AMD Opteron [25] and Intel Nehalem [22] processors with NUMA-based memory hierarchies, high-core count processors like Intel's recently announced 'single-chip cloud computer' [14], purposefully heterogeneous systems like Intel's Larrabee [19] or IBM's Prism [15] processors, or general-purpose (i.e., x86) cores tightly integrated with specialized accelerators, enabled by advances in on-chip interconnection technology [9, 26].

These hardware developments force re-consideration of the design and implementation of the underlying systems software supporting future many-core applications, since management by a single monolithic system and application stack would likely result in limited scalability and unnecessary software complexity. Instead, and in order to address the heterogeneous nature of future many-core systems, this paper presents an outlook in which (1) platforms are partitioned into multiple 'islands' of resources [14, 23], and (2) each island can run its own system and application stacks, customized to better exploit island resouces (e.g., stacks focused on storage vs. communication [1] resources). Island boundaries may be established based on types of cores on multi-ISA platforms (e.g., an island with x86 vs. GPU cores), on their cores' distances from memory modules (e.g., on NUMA architectures) or on coherence domains, or based on the functional semantics of how its cores are used. For instance, an island of cores focused on communication tasks may export a real-time scheduling policy [20], whereas another may export a scheduling policy optimized for server tasks, etc.

'Islands' of cores is not a new notion, in that previous research has already established the utility of partitioning platforms and higher level systems stacks into sets of tiles, clusters, or cells [5, 23, 10, 16, 14], demonstrating that this approach can help improve scalability and isolation. This paper's new contribution, however, is to identify and address two key problems with islands and their independent resource managers:

1. *Maintaining global properties.* With multiple resource managers, it becomes difficult to attain desired platform-level or end-to-end properties. For example, when an application spans multiple islands whose internal resource managers make their own scheduling decisions, how do we provide it with appropriate levels of end-to-end service performance?

2. *Dealing with heterogeneous abstractions.* The managers present in multiple islands will each use and support different sets of resource abstractions, an example being virtual machines or processes supported for

sets of x86 cores and communication queues and messages managed in the ixp network processor. In the presence of such diversity, are there standard communication and coordination interfaces that abstract heterogeneity while still allowing managers to share and act on relevant resource management state?

Next, we first motivate and demonstrate the need for coordinated management, followed by second, a statement of requirements for coordination mechanisms and methods.

Need for management coordination in heterogeneous systems. With reference to problems 1. and 2. above, we next describe compelling use-cases for heterogeneous multicore platforms for which coordination between independent resource managers is an essential feature of future systems software.

1. *Meeting application requirements.* Consider a prototype heterogeneous platform comprised of a general purpose set of x86 processors connected over PCIe to an IXP Network processor [1]. The platform is used to run the RUBiS web application, which is an eBay-like auction website benchmark (see Figure 1). The x86 processors are managed by the Xen hypervisor [4], where RUBiS is run by placing its three major components, namely the Web, Application and database servers, into separate virtual machines. Requests issued by external clients are handled by the IXP platform component, which acts as a programmable network interface that sends and receives RUBiS traffic between our prototype host and clients. Previous work [3, 27, 2] has shown that the resource usage of multi-tier applications is governed by incoming client requests and their types. Exploiting this fact, a request classification engine performing deep packet inspection and running on the IXP processor can be used to better manage the CPU resource allocations given to individual RUBiS components running on the x86 processors. Needless to say, the performance improvements sought in this fashion cannot be realized unless there are well-defined and efficient interfaces between the message-centric resource management methods existing on the IXP (e.g., the priorities used for servicing different message queues) and the process- or VM-centric management methods used on the x86 platforms. This is demonstrated in Figure 2, which shows the minimum and maximum end-to-end response time latencies for various RUBiS request types, as observed by the client in this setup. These measurements show substantial variation in the minimum and maximum response time latencies of requests, which as shown in Section 3, are due to the fact that there is no coordination between the IXP's queue-centric and the x86's VM-centric resource management actions. We also note that there are additional examples that demonstate the need for coordinated resource management, including recent work

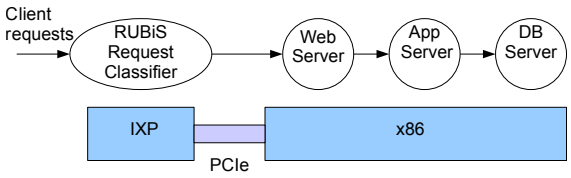

Figure 1: RUBiS Components on IXP and x86 systems and their Interactions on Receive Path.

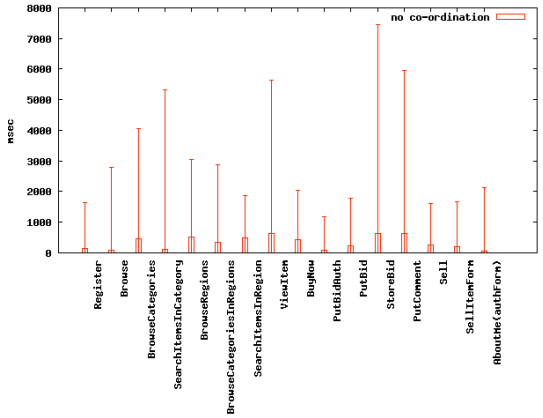

Figure 2: RUBiS: Variation in minimum-maximum response latencies.

in which performance improvements are gained by better co-scheduling tasks on graphics vs. x86 cores to attain desired levels of parallelism [12, 13].

2. *Platform-level power management.* While power budgeting can be performed on a per tile-basis (e.g., in the upcoming Intel chip [14])), it is well-known that properties like caps on total power usage must be obtained at platform level. This is because turning off or slowing down processors in certain tiles may negatively impact the performance of application components executing on others. Maintaining desired global platform properties, therefore, implies the need for coordination mechanisms [17, 28], which at the same time, act to preserve application-level quality of service or performance constraints.

Remainder of paper. The remainder of the paper is organized as follows. Section 2 explains our current implementation of coordination for the prototype heterogeneous platform used in this research. This is followed by experimental evaluations in Section 3 demonstrating the value of coordinated resource management. Section 4 takes a look at related work relevant to our research. Conclusions and future work appear at the end.

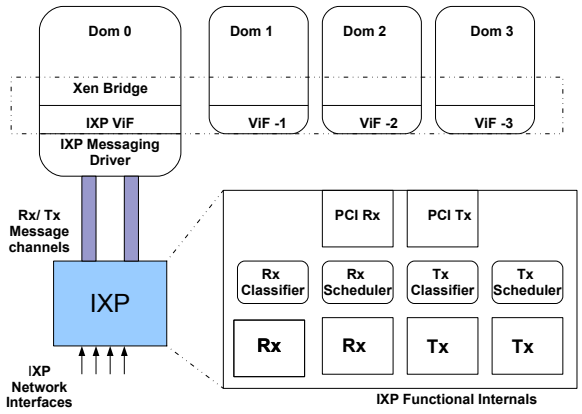

Figure 3: Execution model: x86-IXP prototype.

2 Implementation

Section 1 explained the need for coordination mechanisms in heterogeneous systems. As an example two-island heterogeneous setup, we have developed an experimental prototype using a general purpose x86 platform connected over PCIe to an IXP network processor [1] (see Figure 3). There are two scheduling islands in our setup:

(1) an island consisting of x86 cores, managed by the Xen hypervisor (not shown in the figure for clarity) and the privileged controller domain Dom0, and (2) an island consisting of specialized communication cores on the IXP, managed by the IXP-resident runtime and via a device-driver interface embedded in the Dom0 kernel.

All communication to and from the x86 (i.e., the VMs) is performed via a virtual interface (ViF), implemented on top of a vendor-provided messaging driver. The IXP ViF interfaces with the Linux TCP/IP network stack. It receives packets from the messaging driver interface, converts them to valid socket buffers, and sends them to the kernel network stack. Packet transmission from the host is handled in a similar way. The IXP ViF first converts the socket buffers into valid packet buffers for the messaging driver. These are later dispatched to the IXP via DMA. Using the Xen bridge tools, we make this IXP ViF the primary network interface for network communication between Xen DomUs and the outside world.

2.1 The IXP island of cores.

IXP Architecture. The IXP 2850 used in our research is a programmable network processor with 16 8-way hyper-threaded RISC microengines running at 1.4 GHz clock frequency. The instruction set supported by the microengines is optimized for packet processing-related tasks, thereby making these cores suitable for communications. The platform has a deep memory hierarchy, with increasing access latencies at each level. Closest to each processing core, each microengine has 640 words of local memory and 256 general purpose registers. Next, there are 16KB of shared scratchpad memory, 256 MB of external SRAM (used primarily for packet descriptor queues), and 256MB of external slower DRAM memory (used for packet payload), all of which can be used for inter-microengine communication. The external memories are also mapped into host memory and accessible from the host. In addition, the hardware supports signals, which can be used for inter-thread signaling within a microengine, as well as externally between micro-engines. An ARM XScale core, used for control and management purposes, runs Montavista Linux. Communication with the host is performed via one or more message queues between Dom0 and the IXP. The message queues contain descriptors to locations in a buffer pool region where packet payloads reside. Both, the message queues and the buffer pool region are part of reserved memory in the host physical address space. The buffer pool management and message descriptor transfer on the host side is managed by a messaging driver in the Dom0 kernel. On the IXP end, two micro-engines, labeled PCI-Rx and PCI-Tx in Figure 3, manage the same functions for the IXP processor and IXP DRAM packet rings. The messaging driver handles packet-receive by periodic polling. The IXP can be programmed to interrupt the host at a user-defined frequency. Every time this interrupt is serviced by the messaging driver, the host-IXP message queues are checked for any outstanding descriptors which are then dequeued and passed to upper layers in the network stack.

IXP as a scheduling island. The IXP microengine threads, except for those designated for PCIe-related operations, are programmed to execute one of the following tasks: packet receipt (Rx), packet transmission (Tx), or classification (on the Rx or Tx flows). By default, the scheduling of these threads is round-robin, purely managed by hardware, with context switches occurring on each memory reference. We implement scheduler-like functionality on top of this round-robin switching for the Rx- and Tx-related tasks. These schedulers use in-memory data structures and signals to notify threads to explicitly yield or to start executing, and to schedule the receipt/transmit operations and packet enqueue/dequeue on the IXP-host messaging interface. This helps in achieving 'weighted' scheduling/resource management for packet Rx and Tx operations, where quality of service for classified flows can be managed by tuning the number of threads assigned to each flow. For instance, if the

classification engine classifies incoming packets into per VM flow queues, then by tuning the number of dequeuing threads per queue and their polling intervals, we can control the ingress and egress network bandwidth seen by the VM. Our goal is, then, to coordinate these thread scheduling and queue management actions with the x86 scheduler.

2.2 The x86 island of cores.

The second scheduling island in our x86-IXP prototype consists of an x86 multicore platform, virtualized with the Xen hypervisor. The island's resource management is performed by the Xen credit scheduler and the privileged controller domain, Dom0. Virtual machines upon creation are assigned weights that are translated internally by Xen into credits and are allocated CPU resources in proportion to their weights according to the credit scheduling algorithm [8]. The controller domain hosts a user-space utility 'XenCtrl interface' to tune the credit scheduler behavior and adjust processor allocation to individual guest VMs.

2.3 x86-IXP coordination.

In order to coordinate resource management across the x86 and IXP scheduling islands, we need to identify first, the islands in our system and then, the processes that will execute in one or part of both islands (e.g., the IXP needs to know of guest VMs on the x86 island that will send and receive network traffic through it). At system initialization time, all scheduling islands register with a global controller (i.e., the first privileged domain to boot up and have complete knowledge of the system platform, in our prototype, this function is a part of Xen Dom0). When guest VMs containing application components are deployed across the platform's scheduling islands, they register with Dom0. In this way, identifier information about VMs using the IXP as a network interface will be coordinated with the IXP island through its device driver interface in Dom0. Part of the PCI configuration space of the IXP device is used to setup a coordination channel between the IXP and the x86 host, used for exchanging messages between the two islands which drive various coordination schemes, further discussed in Section 3.

3 Evaluation

We next experimentally demonstrate the feasibility and the importance of coordinating resource management actions across scheduling islands. Experiments are conducted on our x86-IXP prototype described in Section 2. It consists of a Netronome i8000 communications accelerator based on the Intel IXP2850 network processor

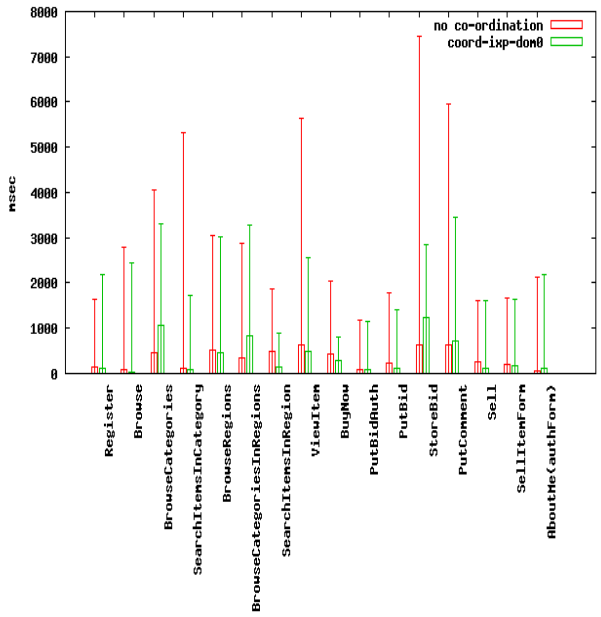

Figure 4: RUBiS Min-Max Response Times. Coordination helps in peak response latency alleviation.

connected via PCIe to a dual-core 2.66GHz Intel Xeon processor. The host processor runs Xen with a Linux 2.6.30 Dom0 kernel. Experimental analyses are conducted using two widely-used benchmarks: (a) RUBiS – a multi-tier auction website modeling eBay, and (b) MPlayer – a media player benchmark.

3.1 RUBiS

The RUBiS setup consists of an Apache web-server frontend, a Tomcat Servlets application server, and a MySQL Database server backend, all deployed in separate Xen hardware virtual machines running Kubuntu 8.04 Hardy 2.6.24 kernel (see Section 1 Figure 1). Each virtual machine is single VCPU and has 256 MB of RAM. Dom0, however, has unpinned VCPUs and can execute on all CPUs. All VMs' network communication is relayed via the Xen bridge interface to the IXP accelerator. The IXP runtime acts as a front-end to all network-related activity of Xen VMs and is responsible for relaying packets to and from the wire and external RUBiS clients to the host. A RUBiS client is deployed on a separate x86 dual-core host, running Kubuntu Hardy 2.6.24 kernel with 384 MB physical RAM. The RUBiS server-side network interfaces and the client interface are on the same network subnet.

Analyses of requests' resource requirements.

We use offline profiles of behavior of the RUBiS components for various workloads to *actuate* coordination. Profiles are based on two client workloads available with the standard RUBiS benchmark: browsing (read) mix

Request Type	Base(ms)	coord-ixp-dom0(ms)
Register	1447	1015
Browse	922	461
BrowseCategories	1896	1242
SearchItems-InCategory	1085	788
BrowseRegions	1491	1490
BrowseCategories-InRegion	1068	927
SearchItems-InRegion	590	530
ViewItem	2147	1944
BuyNow	551	292
PutBidAuth	1089	867
PutBid	1528	538
StoreBid	3366	1421
PutComment	4186	721
Sell	720	490
SellItemForm	351	188
AboutMe(authForm)	1154	546

Table 1: RUBiS - Average Request Response Times.

and bid/browse/sell (read-write) mix. Request traffic from the client follows probabilistic transitions emulating multiple user browsing sessions, and consists of approximately twenty basic request types (see Table 1). Offline profiling establishes relationships between the properties of the incoming request types and the resulting inter-VM communications: (1) for the browsing (read only) mix, static content like HTML pages and images need to be served for the client, resulting in a large amount of webserver-application server interactions, and practically no database server processing; (2) for the bid/browse/sell (read-write) mix, dynamic content using servlets, reads, and writes to and from the backend database generate a large number of application – database server interactions. In addition, the application server utilizes the CPU more heavily, as it is also serving dynamic content by running Java servlets. These observations are consistent with results from previous work [3, 27].

Coordination scheme. Based on insights into the relationships between request types and the resulting component interactions and resource requirements, coordination needs to use the application-level knowledge (about client request types) on the IXP island to possibly change scheduling of the RUBiS VMs in the x86 island. The goal is to maintain the following performance properties:

- *low response-time variability* – end-user experience depends on how 'responsive' the website appears to be, which requires not simply low average response time, but rather a tolerable standard deviation limit across multiple requests of the same type;
- *high request throughput rate* – resulting in higher scalability of the RUBiS server;
- *low average session time* – affecting both end-user experience and server scalability; and
- *high platform efficiency* – a measure of the average request throughput (i.e., application performance) over the mean CPU utilization (i.e., resource utilization), since the use of only a system-level metric like CPU utilization does not provide sufficient insight into how that utilization is translated into better application performance.

To obtain these properties, the IXP scheduling domain requests weight adjustments to be applied to RUBiS VMs in the remote x86 scheduling domain. Browsing related requests result in sending 'weight_increase' messages for the web VM and 'weight_decrease' message for the database server, whereas servlet versions will correspond to 'weight_increase' messages for the database server domains. Given that the application server sees increased activity for processing both request types, its weight is increased in accordance with web server weight for read requests, and with database server weight for write requests.

We compare this coordinated case against the baseline case when there is no coordination across the IXP and x86 scheduling domains.

Benefits of coordination. Experimental results presented in the remainder of this section demonstrate the benefits of coordination for achieving improvements in each of the aforementioned metrics for the RUBiS overlay. Figure 4 shows the min-max response times for serving different RUBiS requests in a read-write browsing mix workload. We observe that the coordinated case results in reduced standard deviation for every request type serviced, sometimes by up to 50%. The use of our coordination results only in slight overheads by increasing the minimum response time latency by up to tolerable 3%.

We do not currently incorporate any mechanisms for predicting frequent transitions amongst read and write requests or to recognize oscillations in client request streams and all our coordination actions are applied on a per-request basis. Another issue is the relatively large latency of the PCIe-based messaging channel in our current prototype. Both combined sometimes lead to the incorrect application of our coordination algorithm when managing resources (e.g., the maximum response time for 'BrowseCategoriesInRegion', a browsing request type is higher for the coordinated case). The correctness of this interpretation of results is demonstrated by another run of a purely "Browsing" related mix that does not have the read-write transitions. Here, our ap-

	Base (req/s)	coord-ixp-dom0 (req/s)
Throughput	68 req/s	95 req/s
Sessions completed	6	11
Avg session Time	103s	73s
Platform Efficiency	51.28	58.20

Table 2: RUBiS – Throughput Results.

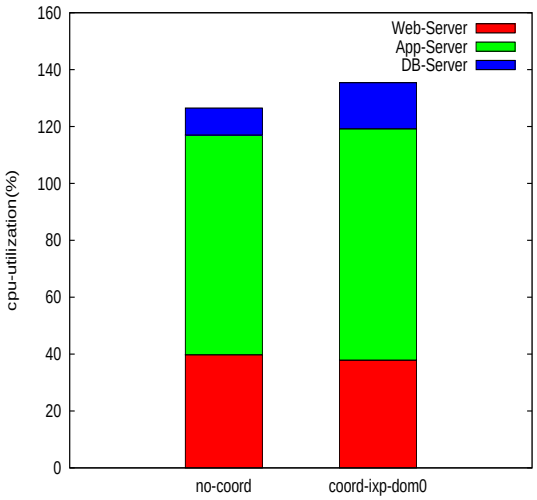

Figure 5: RUBiS CPU Utilization.

proach always performs better than the baseline case for all request types.

The results in Table 1 show a similar trend with respect to the average response times for the same read-write workload. Our coordination algorithm significantly reduces response times for all categories of requests (including by over 60% for 'PutBid' requests), Table 2 shows additional performance metrics for the RUBiS benchmark, where the use of coordination clearly results in improved performance and more efficient utilization of platform resources. Concerning 'raw' resource utilization, Figure 5 shows small increases in CPU utilization in the event of using coordination. These results are gathered for the same read-write request mix as above, for which there is higher application and database server activity, justifying the higher weights (i.e., resource allocations) for these components. We also observe that with coordination, the user space CPU utilization within the guest domain is increased, while iowait and the system CPU utilization values decrease. This is advantageous, as it means the application receives more CPU time to run. The platform efficiency metric in Table 2 justifies the resulting higher CPU utilization with a larger improvement in application performance (e.g., throughput), thus demonstrating the importance of coordinated resource management.

3.2 MPlayer Benchmark

Mplayer is an open-source movie player benchmark. It plays most video formats and supports a variety of codecs including the h.264 high definition codec. Retrieving video streams and playing them requires decoding the codec used by the stream. This is a fairly high CPU-intensive task. The amount of CPU usage necessary to provide a desired viewing experience depends on certain stream characteristics, such as the type of codec, resolution, frame- and bit-rate. Higher bit-rate and higher frame-rate video guarantees better video quality and smoother viewing. However, decoding these streams is more CPU-intensive.

We use the IXP-based testbed with two Mplayer clients inside two virtual machines, both 256 MB, single VCPU, running Kubuntu 8.04 2.6.24 kernel. A Darwin Quicktime streaming server is deployed on an external machine, serving video streams over RTSP and UDP. All network communication between the client and server is directed through the IXP interface. The IXP processor classifies incoming streams based on virtual machine IP address that hosts the MPlayer client. Mplayer supports a benchmark option that plays out the streams at the fastest frame rate possible and we also disable video output for all our tests, just focusing on the decoded frames/sec output as our application-level quality of service metric.

Coordination schemes.

1. Using application knowledge. In order to drive coordination, we devise a coordination scheme that leverages the incoming stream properties and hence application knowledge to drive coordination between the IXP and the x86 scheduling domain. To do this, when an RTSP session is established, the IXP maintains bit- and frame-rate state on a per guest virtual machine basis that hosts the MPlayer client. The actual incoming stream is classified based on the destination (i.e., guest) IP address. The IXP sends an 'Increase_weight' message for a high bit-rate, high frame-rate stream, whereas 'Decrease_weight' message is sent when servicing low bit-rate, low frame-rate streams. The results in Figure 6 show that this coordination results in an improved overall frame rate. In this experiment, we first start the guests with default weights of 256 each. Domain-1 plays a lower frame-rate (20 frames/sec) 300 kbps stream, while Domain-2 plays a higher frame-rate (25 frames/sec) 1Mbit stream. With default weights, neither guest domain is able to meet the required frame-rate guarantees. When we increase their weights due to their high bit-rate detection, Domains 1 and 2 report output frame rates of 22 and 25.7 frames/sec,

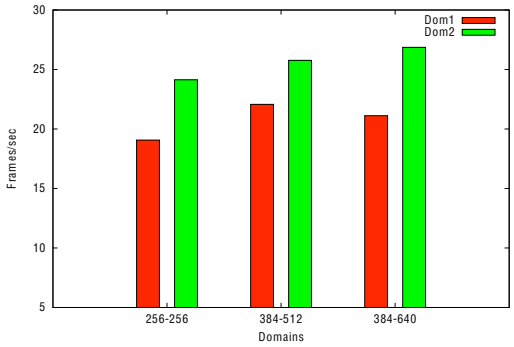

Figure 6: Mplayer: Video-stream Quality of Service.

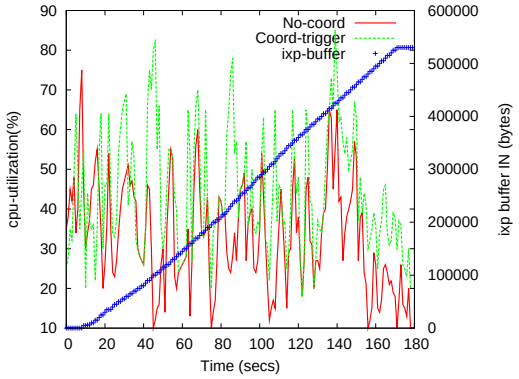

Figure 7: Mplayer: Tuning Credit Adjustments using IXP Buffer Monitoring.

Guest Domain	Baseline Frames/s	With Co-ord Frames/s	% change
Domain-1	24.0	26.6	+9.77
Domain-2	80.0	75.0	-6.25

Table 3: MPlayer – Trigger Interference.

respectively, which means that both meet their required frame-rate values. If we now further increase Domain-2's weight because it has a higher frame-rate requirement and also increase the number of IXP threads servicing Domain-2 receive queue in tandem, we see that Domain-2 achieves still better frame-rates, but Domain-1's frame rate is reduced in proportion to Domain-2's increased weight. It still remains above the 20 frames/sec limit, however. Hence coordination helps us to translate stream-level properties into appropriate CPU resource allocation for MPlayer.

2. *Using system buffer monitoring.* In the previous example, we apply higher-level application properties to drive coordination. We next actuate coordination, which does not rely on application-level knowledge but solely system-level monitoring insights. To demonstrate such a use-case and its benefits, we monitor network-buffer lengths in the IXP DRAM which correspond to packet queues for the host VMs. If the packet-rate increases like it may for streaming applications (e.g., in UDP bulk transfers with no flow-control), such a change will be noticed at the first stage of the pipeline – the IXP scheduling domain. This information can be used to inform later stages that they will need additional processing power, thereby anticipating or avoiding potential bottlenecks. Such actions are time-critical because if not dequeued in time, the frontend buffer could overflow, leading to lost packets. In our streaming case, whenever the buffer-length goes above a defined threshold, an immediate trigger notification is sent to the x86 host, which should boost the dequeuing guest VM's position in the runqueue. As can be seen from Figure 7, we see spikes in CPU utilization for the boosted domain whenever a buffer-threshold of 128Kbytes is reached on the IXP. The outcome is an increased frame-rate of 26.6 frames/sec as compared to the baseline case of 24.0 frames/sec – a near 10% improvement.

Trigger overheads. Finally, we also evaluate the impact of such trigger coordinations on other VMs running in the target scheduling island. Towards this end, we deploy a second MPlayer VM – Dom-2, which rather than playing video from the network, plays it from its own local disk. Therefore, this VM does not use any resources of the IXP island. Our measurements, shown in Table 3, show that Dom-2's performance degrades by only 6%. While there is still an overall net gain in platform efficiency, we believe that on more tightly coupled next generation manycores, the overheads generated by such triggers will be substantially reduced.

3.3 Discussion of results – a case for coordination.

From the experimental evaluations described above, we observe that coordination helps improve the performance of applications spanning heterogeneous islands, and it also leads to more efficient use of platform resources (i.e., through increased CPU usage efficiency). Given these encouraging results, we believe that there exists a set of low-level coordination mechanisms that should be an essential feature in the design of future system software for heterogeneous manycore platforms. For the co-

ordination schemes in our evaluation use-cases, we identify two such mechanisms that can effectively be used to implement more versatile and complex higher-level coordination algorithms.

1. Tune – is a mechanism used by an island to request fine-grained resource adjustment of a particular entity (e.g., process or VM) in a remote island. This mechanism corresponds to the 'Weight_increase' and 'Weight_decrease' messages used in our evaluated coordination algorithms. Messages containing a process or VM identifier and a +/- numerical value can be used to request resource adjustment that, at the remote island, will get translated into corresponding weight or priority adjustments, depending on the remote island's scheduling algorithm (e.g., credit adjustments in Xen scheduler or poll time adjustments in an I/O scheduler).

2. Trigger – is an immediate notification, like an interrupt between two islands. It is a mechanism that lets an island request resource allocation for a particular process in a remote island as soon as possible, and therefore has more preemptive semantics.

Hardware considerations. Future platforms [14] may have non-cache coherent memory between islands of cores, and therefore supporting these mechanisms via message-based communication [5] contributes to the generality of the approach.

In addition, although we discuss the integration of such coordination mechanisms into system software for heterogeneous many-cores, their realizations can be made more efficient through use of adequate hardware support. First, by leveraging advanced interconnection technologies (e.g., QPI, HTX), more tighly coupled heterogeneous multicores can be realized, which will eliminate the latency concerns, as observed in our experiments due to the use of a PCIe interconnect.

Next, inter-processor interrupts and, for some architectures, the monitor and mwait instructions, are the only relatively primitive inter-core communication methods present in current platforms. The presence of fact core-core hardware-level signalling support, which can also carry the small additional amounts of information as required by the coordination mechanisms described above, can further eliminate some of the observed software overheads.

Finally, use of hardware-supported queues, or use of fast on-chip shared memory with explicit message passing semantics [14] for the inter-island coordination channels can result in improved performance and scalability of such mechanisms.

4 Related Work

Scheduling islands. The concept of scheduling islands introduced in Helios [23] has its roots in earlier work that includes Cellular-Disco [10], Hive [7], K42 [16], and [6]. While Helios uses satellite kernels to build distributed systems in the small and has a notion of heterogeneous runtimes, the Hive system uses resource-partitions for fault-containment, and K42 uses them to exploit locality. The implementation of scheduling islands via virtual machines used in this paper is similar to the approach followed in Cellular-Disco, which uses virtual machines to run as domains in 'cell' partitions. We wish to extend the notion of islands by encouraging coordination mechanisms to be exported directly at the system software layer for better platform resource management, something we believe has not been looked at in previous work.

Concerning scheduler coordination, there is recent work on scheduler optimizations that enhance I/O performance in virtualized environments [11, 24]. Optimizations are obtained by coordinating VCPU scheduling with virtual machine I/O, but the solutions provided rely on a centralized controller domain (Dom0) to provide the scheduler with necessary hints. With our proposed coordination mechanisms we wish to distribute such control across scheduling domains. Further, we explore more complex and richer relationships across multiple domains, based on application-level data flow and control dependences.

Application monitoring. Some of our coordination policy models in Section 3 use application-level dependencies to drive coordination. However application profiling to discover these component dependencies during runtime is not a part of our current work, and so we rely on previous research and our own offline profiling to learn them. For instance, for one of our multi-tier benchmarks, RUBiS, we use insight from previous work [3, 27] to understand the work-flow in such applications based on incoming requests and then use this understanding to drive coordination. Other research conducted in our own group and elsewhere [2, 18], has developed methods for automated discovery of inter-component dependencies in large scale distributed applications, which can be used in conjunction with our coordination schemes.

5 Conclusions and Future Work

This paper presents a case for coordination in heterogeneous multicore platforms. In order to deal with the increased parallelism and heterogeneity on next generation multicores, we rely on platform partitioning into multiple scheduling islands – sets of resources under the control of a single resource manager. The challenge then is how to maintain global, platform-wide properties and how to deal with the end-to-end SLA requirements of applications deployed across multiple, independently managed domains. Experimental evaluations for web and for multimedia applications using a proto-

type x86-IXP two-island heterogeneous multicore platform demonstrate that coordination methods can help applications achieve their end-to-end SLAs (with increased throughput, more predictable and lower response times). Based on these encouraging results, we argue that coordination between distributed islands on future platforms needs to be exported as a set of standard mechanisms and new interfaces at the system software layer itself. We identify two such mechanisms in this paper.

Our ongoing work concerns exploring additional usecases (e.g., memory, power [21] and I/O coordination policies along with CPU scheduling) to better delineate required mechanisms and their functionality. Also ongoing are evaluations of the scalability of such mechanisms to large-scale multicore platforms, part of which involve the use of distributed coordination algorithms across multiple island resource managers.

References

[1] ADILETTA, M., ROSENBLUTH, M., ET AL. The next generation of intel ixp network processors. *Intel Technology Journal* (2002).

[2] AGARWALA, S., ALEGRE, F., SCHWAN, K., ET AL. E2eprof: Automated end-to-end performance management for enterprise systems. In *DSN* (2007).

[3] BARHAM, P., DONNELLY, A., ISAACS, R., AND MORTIER, R. Using magpie for request extraction and workload modelling. In *OSDI* (2004).

[4] BARHAM, P., DRAGOVIC, B., FRASER, K., ET AL. Xen and the art of virtualization. In *SOSP* (2003).

[5] BAUMANN, A., BARHAM, P., ET AL. The multikernel: a new os architecture for scalable multicore systems. In *SOSP* (2009).

[6] BUTRICO, M., DA SILVA, D., KRIEGER, O., ET AL. Specialized execution environments. *SIGOPS Oper. Syst. Rev.* (2008).

[7] CHAPIN, J., ROSENBLUM, M., DEVINE, S., ET AL. Hive: fault containment for shared-memory multiprocessors. *SIGOPS Oper. Syst. Rev.* (1995).

[8] CHERKASOVA, L., GUPTA, D., AND VAHDAT, A. Comparison of the three cpu schedulers in xen. *SIGMETRICS Perform. Eval. Rev.* (2007).

[9] `http://bit.ly/7fA5sb`. AMD Fusion Processors.

[10] GOVIL, K., TEODOSIU, D., ET AL. Cellular disco: resource management using virtual clusters on shared-memory multiprocessors. *ACM Trans. Comput. Syst.* (2000), 229–262.

[11] GOVINDAN, S., CHOI, J., NATH, A. R., ET AL. Xen and co.: Communication-aware cpu management in consolidated xen-based hosting platforms. *IEEE Transactions on Computers* (2009).

[12] GUPTA, V., GAVRILOVSKA, A., SCHWAN, K., ET AL. Gvim: Gpu-accelerated virtual machines. In *HPCVirt* (2009).

[13] HONG, S., AND KIM, H. An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness. In *ISCA* (2009).

[14] HOWARD1, J., DIGHE1, S., ET AL. A 48-core ia-32 message-passing processor with dvfs in 45nm cmos. *International Solid State Circuits Conference* (2010).

[15] IBM. A wire-speed power processor: 2.3 ghz 45 nm soi with 16 cores and 64 threads. *International Solid State Circuits Conference* (2010).

[16] KRIEGER, O., AUSLANDER, M., ET AL. K42: building a complete operating system. In *EuroSys* (2006).

[17] KUMAR, S., TALWAR, V., ET AL. vmanage: loosely coupled platform and virtualization management in data centers. In *ICAC* (2009).

[18] KUMAR, V., SCHWAN, K., ET AL. A state-space approach to sla based management. In *NOMS* (2008).

[19] `http://bit.ly/5sgX8T`. Larrabee: An x86 many-core architecture for visual computing.

[20] LEE, M., KRISHNAKUMAR, A. S., KRISHNAN, P., SINGH, N., AND YAJNIK, S. Supporting soft real-time tasks in the xen hypervisor. In *VEE* (2010).

[21] NATHUJI, R., SCHWAN, K., ET AL. Vpm tokens: virtual machine-aware power budgeting in datacenters. *Cluster Computing* (2009).

[22] `http://bit.ly/5eNDKy`. Intel Nehalem Processors.

[23] NIGHTINGALE, E. B., HODSON, O., ET AL. Helios: heterogeneous multiprocessing with satellite kernels. In *SOSP* (2009).

[24] ONGARO, D., COX, A. L., AND RIXNER, S. Scheduling i/o in virtual machine monitors. In *VEE* (New York, NY, USA, 2008).

[25] `http://bit.ly/8oe8uU`. AMD Opteron six-core rocessors.

[26] `http://bit.ly/5OYmVP`. Intel QuickPath Interconnect.

[27] STEWART, C., KELLY, T., ZHANG, A., AND SHEN, K. A dollar from 15 cents: cross-platform management for internet services. In *ATC'08: USENIX Annual Technical Conference* (2008).

[28] ZHU, X., YOUNG, D., ET AL. 1000 islands: Integrated capacity and workload management for the next generation data center. In *ICAC* (2008).