Hardware-Assisted Context Management for Accelerator Virtualization: A Case Study with RSA

Ying Gao(✉) and Timothy Sherwood

University of California, Santa Barbara, CA 93106, USA
yinggao@ece.ucsb.edu, sherwood@cs.ucsb.edu

Abstract. The advantages of virtualization, including the ability to migrate, schedule, and manage software processes, continues to drive the demand for hardware and software support. However, the packaging of software state required by virtualization is in direct conflict with the trend toward accelerator-rich architectures where state is distributed between the processor and a set of heterogeneous devices – a problem that is particularly acute in the mobile SoC market. Virtualizing such systems requires that the VMM explicitly manage the internal state of all of the accelerators over which a process's computation may be spread. Public-key crypto engines are particularly problematic because of both the sensitivity of the information that they carry and the long compute times required to complete a single task.

In this paper we examine a set of hardware design approaches to public-key crypto accelerator virtualization and study the trade-off between sharing granularity and management overhead in time and space. Based on observations made during the design of several such systems, we propose a hybrid local-remote scheduling approach that promotes more intelligent decisions during hardware context switches and enables quick and safe state packaging. We find that performance can vary significantly among the examined approaches, and that our new design, with explicit accelerator support for state management and a modicum of scheduling flexibility, can allow highly contended resources to be efficiently shared with only moderate gains in area and power consumption.

1 Introduction

Virtualization has emerged as a common means by which one may share and more optimally utilize underlying physical resources. As custom hardware accelerators are called upon to take significant portions of the workload from traditional CPUs, the state of computing tasks is increasingly spread across a set of highly heterogeneous devices. Effective virtualization of a system with such distributed and heterogeneous memory elements can be extremely complicated as both fine-grained scheduling and the safe management of the underlying hardware state may be required [8,12,13]. For each distinct type of accelerator, the

© Springer International Publishing Switzerland 2016
F. Hannig et al. (Eds.): ARCS 2016, LNCS 9637, pp. 72–83, 2016.
DOI: 10.1007/978-3-319-30695-7_6

virtual machine monitor (VMM) must be aware of what subset of the machine state is critical to maintain correctness, which subset is potentially damaging if leaked to other VMs, and how critical parts of the hardware state can be managed and restored by the interface provided by that accelerator core.

This complexity also comes with a performance and system management cost, specifically in that it leads to an inability to coordinate the accelerators effectively. Switching the context for an accelerator can have a non-negligible cost (driver/OS and driver/device communication, cleanup, power management, etc.) and that cost can be variable based on time. If the VMM is to coordinate the accelerators it must have an accurate view of what resources are free for scheduling and what the costs of scheduling might be. The VMM must either be able to estimate those costs from models, gather them through further communication with the accelerators (which may be then subject to delay due to resource contention), or give up the opportunity for efficient coordinated control.

There are several ways in which a designer may approach this problem. First, they might consider fixed pass-through (e.g. Intel VT-d [9]) where an accelerator is exclusively assigned to one VM, but this exclusive relationship limits sharing. A second approach is for the hypervisor to arbitrate between several VMs with one VM having access at a time, where the hypervisor halts operation of the accelerator and restores it to a known state between guests [17]. This approach requires very little in the way of both additional memory and network communication, but carries a risk of significantly reduced throughput when interruptions cause the loss of interrupted but unfinished work. A third approach is to avoid dropping unfinished tasks, instead storing the intermediate results in memory for future retrieval. This method prevents wasting of allocated timing slots, but might incur heavy data communication [10,11]. A fourth option is to involve the accelerator itself in the alleviation of context switch overhead. If the accelerator is granted some leeway in when the context switch occurs through a modicum of automation inside of a device, smarter switch timing might be possible saving both time and space. This might require an understanding of the computation and a careful re-architecting of the accelerator.

While performance is one important factor, the sharing of state also needs to be completed in a way that is secure. Given the importance of crypto operations, both in performance and security, they are a natural space in which to study accelerator design tradeoffs. To study the impact and suitability of different accelerator virtualization strategies and to provide optimizations for crypto devices, we implement a series of fast modular exponentiation engines. By making minimum changes to the device interface, we enable hardware assisted context management in such a way as to avoid exposing sensitive intermediate results to the upper system and as to involve local scheduling to improve performance. Our experimental results suggest that above certain switching frequencies, the local context switch approaches achieve significantly higher throughput rate than more traditional schemes and thus enable a new level of fine-grain and fair scheduling. The additional area overhead for our baseline and optimized design to implicitly accommodate four VMs is only 36 % and 15 %.

2 Related Work

The management of accelerator-rich architectures is a very active topic of research, but much of the work is focused on application partitioning and fair scheduling, but less with VM-level sharing. HiPPAI [20] alleviates the overheads of system calls and memory access by using a uniform virtual memory addressing model based on IOMMU support and direct user mode access to accelerators. While it is efficient in limiting overheads at the user/kernel boundary, it lacks support in resource sharing. Traditional accelerator scheduling schemes still rely heavily on usage statistics collected from hardware. Pegasus [8] manages accelerators as first class schedulable entities and uses coordinated scheduling methods to align accelerator resource usage with platform-level management. Disengaged scheduling [13] advocates a strategy in which the kernel intercedes between applications and the accelerator on an infrequent basis, with overuse control that guarantees fairness. Some work tackles the management problem by simplifying accelerator/application integration. VEAL [3] proposes a hybrid static-dynamic compilation approach to map a loop to a template for inner loop accelerators. DySER [7] utilizes program phase and integrates a configurable accelerator into specialized data-path to dynamically encode program regions into custom instructions. While these approaches are intelligent in software partitioning and mapping, they fail to take advantage of hardware assistance in resource managing. Some work starts to look into hardware device reusability: CHARM [6] and CAMEL [5] tackle the sharing and management problem mainly by automating composition of accelerator building blocks (ABBs), primarily stateless arithmetic units in ASICs.

Some projects favor managing hardware states implicitly. Task specific access structures(TSAS) [10] inserts a multiplexer as the input of each FF to select between updating its value from the combinational logic or from previously stored data, or simply remaining its value from the last cycle. This scheme takes the majority of the context switch workload within the device and enables fast switching, but at the sacrifice of non-negligible augmented logic and memory. Hardware checkpointing [11] where the hardware states of a device can be stored and be rolled back regarding checkpoint, hold the potential to minimize area overhead wisely. We recognize the value of hardware checkpointing - in fact we extend its role in coordinated resource management: for accelerators like an RSA engine that implements real-time requests, hardware support in context management will be of great help to fast and fine-grained accelerator sharing.

3 Baseline RSA Accelerator Architecture

3.1 Montgomery's Modular Multiplication and Exponentiation

The core computation in an RSA crypto engine is modular exponentiation, consisting of a number of modular multiplications. Montgomery's modular multiplication algorithm [15] employs simply additions, subtractions, and shift operations to avoid expensive divisions. In this paper we work with an extension to

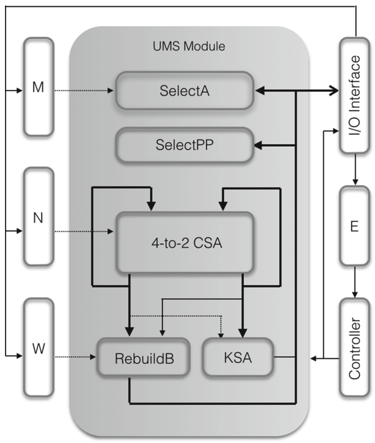

Fig. 1. Traditional RSA accelerator block architecture

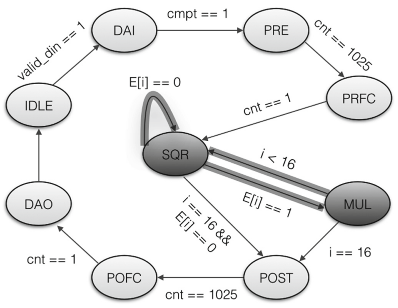

Fig. 2. State diagram of the original RSA accelerator design. PRE/PRFC and POST/POFC are the preprocessing and the post-processing states for domain format and carry-save format conversions. MUL and SQR stand for modular multiplication and square operation respectively.

this algorithm [18]. Three k-bit integers, the modulus N, the multiplicand A and the multiplier B are needed as inputs for computation.

Algorithm MM_UMS is defined as follows:

for i = 0 to i = k − 1 :

$$q = (S + A * B[i])\ mod\ \ 2 \tag{1}$$

$$S = (S + A * B[i] + q * N)\ /\ 2 \tag{2}$$

S is restructured in carry-save form as (Sc, Ss) where Sc and Ss respectively denotes the carry and sum components of S. H-algorithm [2] transforms the computation of modular exponentiation into a sequence of squares and multiplications. Square operation could be performed when both multiplicand and multiplier are identical. The modular exponentiation algorithm, ME_UMS(M, E, N), iteratively applies a unified multiplication or square operation, where for each bit $E[i]$ in exponent E, both a single square operation and multiplication will be performed when $E[i] = 1$ while only a square operation will be performed otherwise.

Figure 1 shows the baseline design. The unified modular multiplication/square module is highlighted in the shadowed region. The nine states in Fig. 2 capture the major stages of the entire modular exponentiation process, as discussed in the algorithm ME_UMS.

3.2 Sharing an RSA Accelerator

One traditional method of device sharing is hard preemptive multitasking. The obvious drawback is that as the switching frequencies increase during heavy sharing, the throughput rate might suffer significant degradation.

To avoid the cost, two options are clear, either the OS relaxes its schedule to wait for the task to complete or the intermediate results from hardware have to be saved for future retrieval. The first option is becoming increasingly difficult since an application can occupy several accelerators simultaneously, thus a perfect point where all devices have just finished their current tasks can be extremely hard to identify or even exist. The latter option seems to comply well with software schedules, but the data movement required to store the intermediate results, coupled with the corresponding memory updates, making this option surprisingly tricky to execute well in practice. Moreover, exposing intermediate results to DMA are also risky due to DMA attacks [19]. A good solution should manage these burdens carefully and a new set of interfaces is needed to simplify the synchronization process.

4 Tightly Integrated Virtual Accelerator Approaches

The simplest tightly integrated design might store all local state in a set of D flip-flops sprinkled throughout the design. However, this approach is also prohibitively expensive. Simulation results suggest that regarding area (and power) efficiency, such virtualized accelerator can add up to a 78 % area overhead.

So what can we do if we want to maintain the accelerator's capability of being fast switched without giving up almost nearly all of our efficiency? We describe two different solutions – the simplest being to replace the local and distributed storage elements with a set of RAMs.

4.1 Baseline Virtual RSA Accelerator Design Overview

In general, most sharing patterns fall into one of the four categories:

- Double Vacancy. No VM is occupying the device.
- Single occupancy. The accelerator is currently dominated by one VM while another VM requires input data streaming for starting a new task.
- Double occupancy. One VM is scheduled to resume a previous uncompleted computation while the other VM is in the process of computing.
- Single occupancy. One VM requires to resume a suspended task while the other VM is performing output data streaming.

Note that in scenario 3, two whole sets of states need to be stored. Based on this, we include 2 KB of RAM alongside the core for temporary storage. We build a simple layer above the RSA accelerator to forward switching commands rather than changing the slave interface directly. We add a *switch* signal underneath the layer to help the controller determine the next state. In order to be able to interrupt a task in the middle of such computation, four more states are added to the FSM. We show the resulting state diagram in Fig. 3.

By enabling hardware preemption, the proposed accelerator virtualization approach successfully realizes the goal of abstracting away hardware details from software without abandoning tasks, at the sacrifice of increasing critical path delay by 16 %.

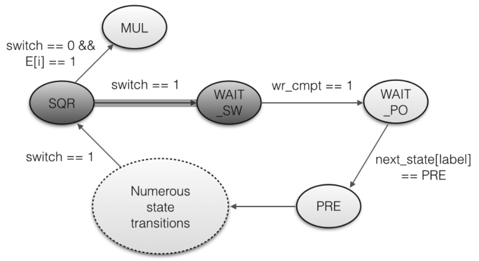

Fig. 3. State diagram of an example transition case in the baseline architecture. When receiving active switch signal in SQR state, it will jump to WAIT_SW state to store intermediate results in local RAM. label denotes VM ID. If the current requesting VM was in PRE state during last switch out, next_state will be set to PRE. After numerous state transitions, the VM that was switched off during SQR state might request again, and have a chance to restore its state

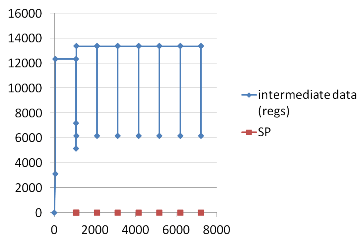

Fig. 4. The amount of local memory needed for storing intermediate results. The y-axis denotes the number of hardware registers and the x-axis denotes timeline measured by clock cycles during a single modular exponentiation operation

4.2 Optimized Accelerator Virtualization Strategy - Making Virtual Accelerator Out of Area Efficiency

In order to eliminate the increased critical path delay, we examine the registers that contain useful intermediate results along the entire process of a single modular exponentiation task. We measure the amount of memory needed to store these intermediate results against execution time cycle Fig. 4.

At the completion of a modular multiplication or a square computation, only the value of Sc and Ss (1025-bit register arrays) are a must-save among all the large register arrays. These transition points, which we informally call SP (*sweetspots*), can be intuitively pinned from the FSM inside the device controller. If we can make sure all switching operations happen at these sweet spots, we can significantly reduce the RAM size required.

To achieve this goal, the device controller is slightly modified to ensure switching always happens at these spots. Upon each major state transition, the contents of Sc and Ss will be forwarded to two designated register arrays Sc_SW and Ss_SW. Note that the contents of these two registers will be refreshed every time an SP is identified and will be flushed during switching operation Fig. 5.

We also want to make sure that the OS gets control of the preemption delay so that it can make scheduling decisions easily when it needs to context switch among a number of concurrent applications. Upon receiving the switching command, the device will compare the time bound to its backward counter and make a decision about whether to reach the next SP or to simply fall back to the last stored one. If the time bound is equal to or smaller than the value of the counter, the current multiplication computation will be abandoned and contents

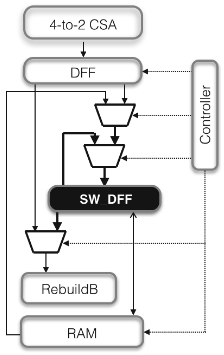

Fig. 5. Design of optimized context switch enforcement in detail. SW_DFF represents register arrays including Sc_SW and Ss_SW storing intermediate results at previous SP

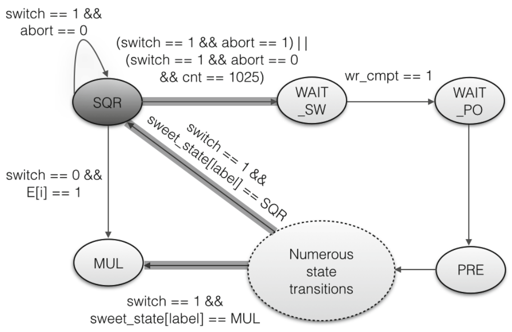
switch == 1 && abort == 0
(switch == 1 && abort == 1) || (switch == 1 && abort == 0 && cnt == 1025)
wr_cmpt == 1
SQR
WAIT _SW
WAIT _PO
switch == 1 && sweet_state[label] == SQR
switch == 0 && E[i] == 1
MUL
Numerous state transitions
PRE
switch == 1 && sweet_state[label] == MUL

Fig. 6. State diagram of an example transition case in optimized design. The abort signal calculated from time bound directs next_state when *switch* == 1. The current task is allowed to finish current square operation when *abort* == 0, sweet_state will be updated to its next square (SQR) or multiplication (MUL) operation judged by $E[i]$ for future state retrieval.

of Sc_SW and Ss_SW will be stored to RAM. An extra state, $SWEET$, is added to allow data transfers between registers and RAMs during task switching. The state that leads to $SWEET$ will be recorded. The relaxed timing bound can be very convenient for scheduling purposes, considering it is difficult for OS to decide the exact best timing to switch in a device. Granted with local scheduling power, the device can wisely help a task fully utilize its time slots. We show an example state transition scenario in Fig. 6 for illustration.

The design removes multiplexer arrays from the critical paths, significantly lowering area cost. Meanwhile responsiveness to interrupts or context switch commands is still guaranteed. Note that these modifications can be generally applied to public-key crypto accelerators. By simplifying the device interfaces, the VMM's scheduling becomes easier and more flexible. Tasks with higher priorities can always be ensured a quick access to hardware acceleration. The hardware accelerator manages to secure itself in a blackbox, without exposing hardware information unnecessarily.

5 Experimental Evaluation

Our evaluations are based on RTL prototypes of accelerators with standard AHB I/O interfaces written in Verilog under the ModelSim [14] environment. We test through the encryption process and use a Verilog testbench with a public exponent 65537 and modulus generated from OpenSSL [16] for encryption. We synthesize all of the RTL designs using the Synopsys Design Compiler [4] with a 45 nm library and collect critical path delays and executing clock frequencies.

The area and peak power models for our embedded memories are based on CACTI 5.3 [1] and for logic and registers based on the results from Design Compiler.

5.1 Relative Performance

One important measure of performance is the total virtualized device throughput as measured by the numbers of encryptions per second. We compare the virtualized throughput rate from each of the designs to the upper bound of performance where each VM is given a completely independent copy of the device (i.e. no interference at all). We simulate three scenarios representing light (concurrently running two VMs while requests from each VM fills 10 % of its timeline), medium (two VMs with 20 % requests) and near-saturating workloads (four VMs with 20 % requests) respectively. The only contention for the crypto accelerator is from multiple VMs attempting to access the engine at the same time. Figure 7a–c depict the relative performance of virtualized devices under these loads respectively.

The y-axis of each of these plots is the relative performance of the different schemes (as compared to our ideal case). The x-axis is the time slice granularities under which the VMs are driving our accelerators. To simulate the fact that one does not switch between VMs instantaneously, a running task will not attempt to switch in a period smaller than a defined time slice. In real-time, latency sensitive, or reactive systems a design may be called upon to switch very quickly. To quantify the suitability of each of the previously described accelerator virtualization under various different switching speeds, we inject requests with the size of one task (for us, the crypto operation) but constrain the minimum window under which those switches might occur. The courser the minimum time between switches, the less we would expect to lose in wasted cycles as computation is abandoned on a switch, but more applications will have to wait to get their computation onto the accelerator. On each of the graphs there are 3 different bars labeled “Tra” for “Traditional” which drops unfinished tasks on a switch. “Base” saves all of the hardware state as described in Sect. 4.1. Finally, “Opti” adds the hardware necessary to allow the accelerator to delay the switching under a fixed bound as described in Sect. 4.2.

As can be seen in these graphs, when the request workload is comparatively low, the performance disparities among the three approaches are not as significant as those when task workload is heavier. However, the performance of the optimized design is consistently the highest throughout all the switching frequencies simulated. The base design has a slight advantage over traditional design when the time slice is smaller than the time for one encryption operation. The advantage more fully manifests when the amount of requests increases. The performance of all three approaches in all the scenarios reaches a peak around and slightly above 25 μs time slice. However, when we compare 25 μs to 100 μs granularities of the three figures, we can clearly see that the peak period tends to shrink as the workload increases. When reaching a comparatively coarse grain

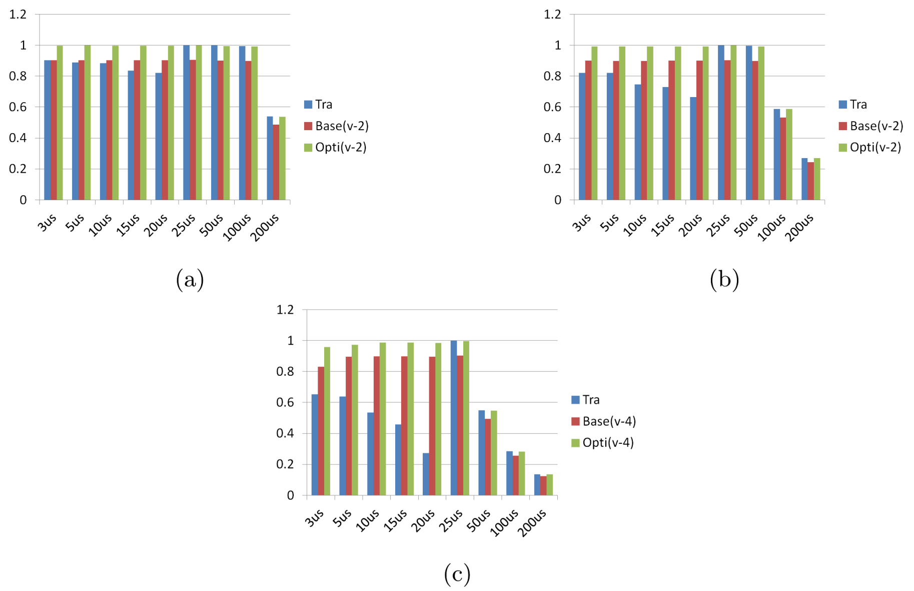

Fig. 7. Relative performance under light (a), medium (b) and near-saturating (c) workload scenarios. V-2 and v-4 denotes the default maximum number of VMs allowed to concurrently occupy the device.

scenario around 200 μs the performance of all of the virtual devices suffer significantly. In these situations the bounds on switching time is large enough to cause a significant amount of idle time in the hardware. The optimized design outperforms the baseline consistently because the more restricted save points limit the hardware needed and the longer paths they cause.

One interesting observation is the non-monotonic performance of the traditional design. The throughput rate drops as the switching frequency rises until it reaches around 1/25 μs. The reason behind this pattern is that when a task is switched off and dropped, the device is more likely to waste more computation cycles when the device is only allowed to be switched at a granularity slightly smaller than time of one operation. Provided that the v-4 optimized design shows at most a 3.6X performance improvement compared to the traditional design, in 20 % workload scenario and reliably high efficiency throughout fine-grain granularities, the optimized design appears to be a clear choice in systems requiring very fine-grain switching when we consider performance alone.

5.2 Area Cost and Power Consumption

To model the area overhead and power consumption of the three virtual accelerators, we synthesized our RTL design in the TSMC 45 nm technology. Results show that the original accelerator occupies 0.11 mm^2 with a peak power consumption of 54.7 mW at 1.6 GHz. Due to the lack of publicly available SRAM

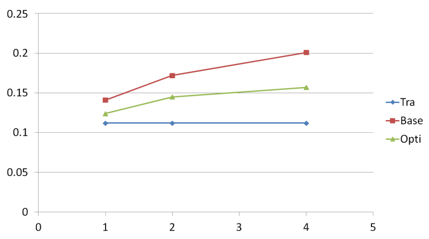

Fig. 8. Comparison of area costs for v-1, v-2 and v-4 designs

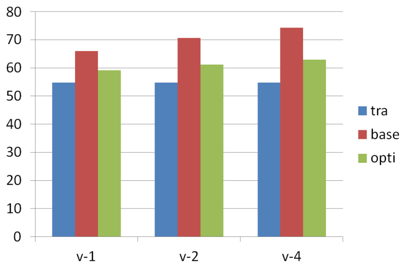

Fig. 9. Comparison of peak power consumption for v-1, v-2 and v-4 designs

compilers in this technology, we use CACTI 5.3 [1] to estimate the area and power of RAMs.

The area cost is shown in Fig. 8. The y-axis of the area plot is the absolute area costs measured in mm^2 unit of the different schemes. The x-axis is the number bounds of VMs that are allowed to be running concurrently on the accelerator (e.g. 2 corresponds to v-2 design). As we can see in the graph, the additional area overhead of the baseline design compared to the traditional design can increase area by up to 29 % for v-2 and up to 36 % for v-4. This extra price paid is primarily due to the additional arrays of multiplexers needed to switch between states and the additional RAMs needed to store the contents of all registers. Note that the optimized design scales better than the baseline. The area overhead is merely 12 % and 15 % for v-2 and v-4 respectively.

Similar to the area costs trends, plots of the peak power consumption present an increasing pattern somewhat proportional to area costs. As we can see from Fig. 9, where the y-axis denotes the absolute peak power consumption measured in mW unit of the different devices, the optimized design scales better from v-1 to v-4 than the baseline design as the default bounds of running VMs increase. The traditional design stands out due to its more uniform power consumption.

An important conclusion is that the baseline design performs slightly worse than both the traditional and the optimized design regarding power consumption, whereas the traditional one suffers significantly reduced throughput/watt rate for near saturating workloads when the time slice is small. While baseline and optimized design already provide with most responsiveness, it is as well likely that energy consumption can be compensated from simplified software level synchronization. Moreover, the internal memory read/write structure guarantees a quick and safe access to intermediate data without dealing with I/O hazards.

Due to its performance and power-friendly benefits, the optimized design improves the throughput/watt rate by at most 3.1X over traditional design when above switching frequency of 45 KHz magnitude and remains competitive to traditional design throughout all sharing granularity range under examination.

6 Conclusions

Growing heterogeneity in hardware devices continues to put easy and safe management in direct conflict with fine-grain scheduling and virtualization. Rather than take a top-down approach requiring that all accelerators be implemented in a particular style, we take a bottom up approach, looking at what it takes to manage the state of a device. In particular we found that there is a small but non-negligible penalty for adding in explicit access to the accelerator state both in terms of area and power. However, we also observe that there is an interesting and previously unexplored trade-off between the scheduling power one imbues the accelerator with and the efficiency with which the schedule can be managed to minimize the waste of timing slots.

With that said, under these limitations we presented comparisons of three different accelerator virtualization schemes working to manage a critical device - an RSA accelerator. When a high degree of sharing and switching is required, the traditional task-dropping scheme can suffer significant performance degradation. If such conditions are expected, a hardware preemption scheme can be adopted, and with a bit of analysis, is able to alleviate the burden of resource scheduling and context management, and to prevent sensitive intermediate data exposure. Results show that our proposed approach manages to dramatically diminish the performance degradation of the traditional scheme and to compensate a naive TSAS in a low-overhead manner both in area and power.

References

1. 5.3, C.: http://quid.hpl.hp.com:9081/cacti
2. Chen, J.H., Wu, H.S., Shieh, M.D., Lin, W.C.: A new montgomery modular multiplication algorithm and its vlsi design for rsa cryptosystem. In: IEEE International Symposium on Circuits and Systems, ISCAS 2007, pp. 3780–3783. IEEE (2007)
3. Clark, N., Hormati, A., Mahlke, S.: Veal: Virtualized execution accelerator for loops. In: 35th International Symposium on Computer Architecture, ISCA 2008, pp. 389–400. IEEE (2008)
4. Compiler, D.: https://www.synopsys.com/tools/implementation/rtlsynthesis
5. Cong, J., Ghodrat, M.A., Gill, M., Grigorian, B., Huang, H., Reinman, G.: Composable accelerator-rich microprocessor enhanced for adaptivity and longevity. In: 2013 IEEE International Symposium on Low Power Electronics and Design (ISLPED), pp. 305–310. IEEE (2013)
6. Cong, J., Ghodrat, M.A., Gill, M., Grigorian, B., Reinman, G.: Charm: a composable heterogeneous accelerator-rich microprocessor. In: Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design, pp. 379–384. ACM (2012)
7. Govindaraju, V., Ho, C.H., Sankaralingam, K.: Dynamically specialized datapaths for energy efficient computing. In: 2011 IEEE 17th International Symposium on High Performance Computer Architecture (HPCA), pp. 503–514. IEEE (2011)
8. Gupta, V., Schwan, K., Tolia, N., Talwar, V., Ranganathan, P.: Pegasus: Coordinated scheduling for virtualized accelerator-based systems. In: 2011 USENIX Annual Technical Conference (USENIX ATC 2011), p. 31 (2011)

9. Hiremane, R.: Intel virtualization technology for directed i/o(intel vt-d). Technology@ Intel Magazine 4(10) (2007)
10. Jovanovic, S., Tanougast, C., Weber, S.: A hardware preemptive multitasking mechanism based on scan-path register structure for fpga-based reconfigurable systems. In: Second NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2007, pp. 358–364. IEEE (2007)
11. Koch, D., Haubelt, C., Teich, J.: Efficient hardware checkpointing: concepts, overhead analysis, and implementation. In: Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays, pp. 188–196. ACM (2007)
12. Liu, J., Abali, B.: Virtualization polling engine (vpe): using dedicated cpu cores to accelerate i/o virtualization. In: Proceedings of the 23rd International Conference on Supercomputing, pp. 225–234. ACM (2009)
13. Menychtas, K., Shen, K., Scott, M.L.: Disengaged scheduling for fair, protected access to fast computational accelerators. In: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 301–316. ACM (2014)
14. ModelSim: http://www.mentor.com/products/fv/modelsim
15. Montgomery, P.L.: Modular multiplication without trial division. Math. Comput. **44**(170), 519–521 (1985)
16. OpenSSL: https://www.openssl.org
17. Rupnow, K., Fu, W., Compton, K.: Block, drop or roll (back): Alternative preemption methods for rh multi-tasking. In: 17th IEEE Symposium on Field Programmable Custom Computing Machines, FCCM 2009, pp. 63–70. IEEE (2009)
18. Shieh, M.D., Chen, J.H., Wu, H.H., Lin, W.C.: A new modular exponentiation architecture for efficient design of rsa cryptosystem. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **16**(9), 1151–1161 (2008)
19. Stewin, P., Bystrov, I.: Understanding DMA malware. In: Flegel, U., Markatos, E., Robertson, W. (eds.) DIMVA 2012. LNCS, vol. 7591, pp. 21–41. Springer, Heidelberg (2013)
20. Stillwell, P.M., Chadha, V., Tickoo, O., Zhang, S., Illikkal, R.,Iyer, R., Newell, D.: Hippai: high performance portable accelerator interface for socs. In: 2009 International Conference on High Performance Computing (HiPC), pp.109–118. IEEE (2009)