

Hardware/System Support for Four Economic Models for Many Core Computing

Joseph Sloan and Rakesh Kumar

Coordinated Science Laboratory
1308 West Main St
Urbana, IL 61801

Abstract

This paper argues for a new set of economic models for many-core computing. Current economic models for processors require a customer to estimate her average-case or worst-case computational requirements beforehand and then buy a processor/system that meets those needs. Physically changing the chip is the only way to transition between different peak computational capabilities, but it has associated cost and management overheads. Therefore, users tend to continue using the chip till a new system is bought. This causes computational resources on chips to often not match computational requirements of customers and/or applications, thereby resulting in processors that are often underutilized or over-utilized. Similarly, the one-to-one mapping between a chip and its available peak computational capability that the current economic models assume results in a mismatch between computational requirements of a user and the computational capabilities of a chip. This mismatch will only increase with increasing number of cores on the processor die.

This paper presents four alternative economic models for many-core computing. The proposed models recognize that when a many-core chip is bought, the customer may often wish to pay for less number of cores than what is present on a chip. The models provide flexibility to the customer to change the available peak computational capability of the chip (or effective number of cores) during the lifetime of the chip. Two of the proposed models also allow the user to treat processing as a service by allowing the user to rent computational capability on the chip as well as pay only for the capability used.

We discuss the hardware techniques that are required to support and enforce these economic models while honoring security and privacy concerns. We show that the required hardware support has a small area and power overhead.

1 Introduction

As technological and economic realities keep fueling the current drive toward multi-core architectures, it is likely that very soon we will have processors that consist of tens or hundreds of cores on the same die. Intel already has a chip consisting of 80 cores [5]. Similarly, some network processors already have 200-core architectures [3].

While the architectures may support increasing computational capabilities, different applications have different amounts of thread-level parallelism (as well as different computational requirements). For example, while some applications are highly parallelizable and may be able to use all the cores on a many-core processor (that consists of

⁰University of Illinois at Urbana-Champaign Center for Reliable and High-Performance Computing Technical Report number CRHC-07-07

tens or hundreds of cores on the same die), other applications may have a limited amount of parallelism and may be able to use only a small number of cores on the processor die.

Current economic models for multi-core processors require a customer to estimate her average-case or worst-case computational requirements beforehand and then buy a processor/system that meets those needs. For example, a desktop customer has an option to buy a 160-dollar uniprocessor or a 900-dollar quad-core processor [14, 1, 9] based on her needs. The commitment to a particular kind/type/class of processor (consisting of a certain number of cores) is usually made one time and switching to a different computational capability requires switching the processing chip (and probably the motherboard, etc.).

Physically changing the chip as the only way to transition between different computational capabilities in the current economic models has associated cost and management overheads. For example, every time a customer buys a new chip, she not only pays for a new piece of silicon, but also pays for the cost of packaging (which can be more than the cost of the chip itself [11]). Similarly, physically switching between chips may be too complicated for an average computer user. Therefore, users tend to continue using the chip till the entire system gets upgraded (or a new system is bought). This causes the computational resources on chips to often not match computational requirements of customers and/or applications, thereby resulting in processors that are often underutilized (resulting in “wasted” dollars) or over-utilized (resulting in reduced efficiency which itself may translate into higher overall costs). The problem gets exacerbated as the number of cores on a processor increase.

There is another way in which current economic models prevent computational capabilities of chips from being matched to the computational requirements. They assume a one-to-one mapping between a chip and its available peak computational capability. This means that only fixed, small set of peak computational requirements are targeted for any given market segment as only a small, fixed number of designs are available in any given market segment (Figure 1). Manufacturing more than a certain number of unique designs for a given market segment becomes prohibitively expensive for processor vendors. This is because of increasing NRE costs as well as diminishing profit margins. Since a chip design typically targets only a certain peak computational capability, the mismatch between requirements and capabilities will only increase with increasing number of cores on the processor die, again resulting in wasted dollars and/or reduced efficiency.

This paper presents four alternative economic models for many-core computing. The proposed models recognize that when a many-core chip is bought, the customer may often wish to pay for less number of cores than what is present on a chip. The models provide the customer the flexibility to change the available peak computational capability of the chip (or effective number of cores) during the lifetime of the chip. The *UpgradesOnly* model allows the user to adapt to increased average computation needs by scaling up any time post-fabrication and post-purchase. The *Limited Up/Downgrade* model allows the user to adapt to increase/decrease in computational needs by upgrading/downgrading the chip on demand. The *CoresOnRent* model allows the user to rent access to cores and use the cores as long as the

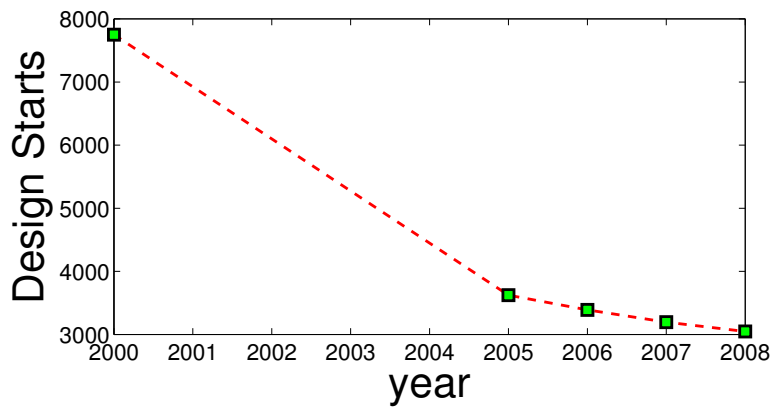


Figure 1. Number of unique ASIC Design starts over time [7]. Overhead costs and increasing complexity associated with new processor designs are driving the number of new processors down every year.

hardware has valid lease. Access to cores is revoked upon lease expiration. The *PayPerUse* model measures and records hardware usage and the billing happens over specified lease period. The last two models allow the user to treat processing as a service by decoupling payment from ownership. Note that these models are in addition to an *IntelligentBaseline* model where the user makes onetime decision about the number of cores she needs and the vendor appropriately enables a subset of cores on the chip before shipping.

We also discuss the hardware techniques that are required to support and enforce these economic models. Hardware support is needed for securely verifying the activation code used to switch between computational capabilities (note that the user is not trusted) as well as for securely isolating the un-licensed/un-purchased cores. We show that the additional hardware that we propose to support and enforce the models has a small area and power overhead.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 outlines the four economic models for many core computing. Architectural frameworks required to support and enforce the proposed models are introduced in Section 4. Section 5 describes specific hardware support to implement these frameworks. We present an evaluation of the overheads for the proposed hardware support in Section 6. Section 7 concludes.

2 Related Work

The economic models that we propose in this paper have been used in several other domains. Cellphone providers, for example, often use variants of the proposed models [2]. Among several reasons for their usage of these models, one of them is the reduced profit margins on the cellphone hardware (one can often get cellphones for free or at a deeply discounted price when one signs up with a cellphone service provider). Similarly, the set-top boxes (or cable boxes) often have new features enabled on demand [6]. Again, the model exists because of relatively high overhead for supporting unique designs corresponding to each combination of features. Our proposal is also driven by reduced

profit margins on hardware, especially per core as the number of cores increase, as well as by high NRE cost for having more than a threshold number of unique designs.

Large scale server systems (or mainframes) also use a *capacity on demand* (COD) model [8, 4, 15] where the economic models are similar. IBM COD systems [8], for example, support models that closely resemble the models that we propose in this paper for many-core computing. Similarly, Burroughs Corporation (which later became Unisys) had mainframes [4] that could be upgraded by enabling a control register. The primary difference between our proposal and the COD and other similar systems is the nature of customers. For the COD systems, the customers are typically relatively small number of trustworthy, visible companies who can always be tracked and made accountable if they do not subscribe fairly to the economic model. With many-core processors, authentication becomes a concern due sheer volume and anonymity of customers. Similarly, security and robustness become a concern as anonymity may encourage malicious users/customers to unlock the processing power. Also, reversibility (or downgrades) needs to be enforced more stringently as there are less ways to force the customer to give up compute resources. As we will see in next few sections, this difference in the nature of customers translates into unique support and enforcement mechanisms.

Having more peak computational capability than what was being exposed to the customer is not an unknown strategy. Radeon 9500 [13] had 8 rendering pipelines even though only 4 were exposed to the user. Similarly, Intel's hypethreading technology was disabled on some processors (e.g., versions of Pentium4) [27]. Our work proposes to expose different capabilities to the user based on different payment amounts and use that to support different economic models.

Using the same *intended* design to target the different power/performance points has been well-known approach. For example, processors are binned into different price ranges based on their operating frequencies as well as power consumption. The variations in power and frequency characteristics for the same intended design are because of within-die and across-die [19, 29] variabilities. Our work differs primarily due to the fact that our intended design is an architecture that supports variable and flexible pricing.

Processors are also often shipped with similar baseline design with minor architectural modifications (also called SKUs). For example, Pentium [10] had versions with 4MB L3, 8MB L3, and 16MB L3 all selling at different prices. Our work is different because we advocate switching between different peak computational capabilities without physically switching the chip.

There has been a lot of work [16, 30, 23, 35, 20] on techniques for isolating regions of a chip for testability and fault-tolerance reasons. Our work uses some of those techniques for switching between computational capabilities. Our architecture needs to be significantly richer, however, to provide authentication, security, and robustness.

Techniques that have been proposed for IP protection [17, 36] and hardware metering [24, 25] are also relevant.

Finally, the proposed economic models have been used in one form or the other for a long time in the software domain [34]. Our proposal is to adapt those models for many-core computing.

3 Economic Models for Many-core Computing

As discussed in Section 1, current economic models result in a mismatch between the computational requirements of a user and the computational capabilities of the chip being used. This results in over-utilization or under-utilization of the processor which, in turn, translates into “wasted dollars”.

We propose four economic models for many-core computing. These economic models decouple the available peak computational capability of a many-core processor from the number of the cores on the processor chip. The models assume that the many-core processor has support for selectively locking/unlocking a subset of cores. Only the cores that are unlocked can be used by the customer. When a customer makes the initial purchase, the vendor selectively unlocks a subset of cores on the many-core chip to match the computational capability being purchased by the customer. The exact hardware mechanisms needed to provide such enforcement are described in Section 5. Below we describe the higher level details for the four models.

- **UpgradesOnly** This economic model recognizes the well-known fact that the *average* computation requirement of a user/customer tends to increase over time. Currently, increased requirements call for system/processor upgrades. The *UpgradesOnly* model assumes that when a customer buys a chip, she picks a chip with the computational capability that is closely matched to her current computation requirements. Therefore, buying a certain computational capability corresponds to the vendor unlocking an appropriate subset of cores. The chip, however, provides user the flexibility to switch to a higher number of cores anytime during the processor lifetime. Purchasing additional cores involves contacting a third party (the hardware vendor, for example) that authenticates the chip and sends activation codes that are used to unlock additional cores thereby increasing the available peak computational capability.

Note that in the *UpgradesOnly* model, the user *owns* the computational capability (or number of cores) that she pays for. We will distinguish this from the *renting* model that we discuss later.

- **Limited Up/Downgrade** This model recognizes the fact that *average* computational needs may sometimes increase only temporarily. For example, the machine that hosts the web server for ISCA-2008 may have significantly higher computational needs close to the paper submission deadline than after the conference is over. Alternatively, there may be scenarios where the currently available peak computational capability is more than the current computational requirements. The *Limited Up/Downgrade* model allows the user to scale up as well as scale back available peak computational capability through processor upgrades as well as downgrades. Both

upgrades and downgrades require contacting the vendor that provides activation codes for upgrades and requires authentication for downgrades. Downgrades involve locking a subset of the currently available cores. Sufficient incentives need to be provided to the customer to request a downgrade. For example, it may involve the vendor providing a refund to the customer. The flexibility that this economic model provides for the customer can potentially increase the customer base for a particular peak computational capability.

Note that in the *Limited Up/Downgrade* model, the user *owns* the computational capability (or number of cores) that she pays for. Downgrades simply represent negative payments.

- **CoresOnRent** This economic model recognizes that there are computing environments where computational requirements are not stable and fluctuate significantly even over a short period of time (of the order of months). In such environments, it may be more reasonable for the user to rent a certain computational capability (or number of cores) than own it.

The model involves the user contacting the vendor for securing access to a certain number of cores for a specified time. The vendor grants the customer a “lease” to the requested number of cores. When the customer’s lease on the hardware has expired, the system automatically locks the leased cores thereby downgrading the peak computational capability. The customer at this point may choose to re-rent the same or different number of cores.

As opposed to the previous two models, the customer does not have ownership of cores (or a computational capability) in the *CoresOnRent* model. In that respect, the chip can be thought of a datacenter where the cycles are being rented [34]. Also, as will be described in the next section, the implementation of *CoresOnRent* has more overhead for the vendor, as well as for the chip than the previous two models.

- **PayPerUse** While the previous models allow the user to buy/rent computational capability based on her estimate of her current (and future) computational requirements, the burden is still on the user to perform accurate estimation proactively. Also, the previous models require the user to contact the vendor for upgrading/downgrading her computational capability.

The *PayPerUse* model imposes no such restriction or requirement on the user. Hardware has support for recording the usage of the chip. This information is transmitted to the vendor which bills the user based on her usage statistics. The vendor activates/unlocks the cores for the next billing cycle upon receipt of the payment from the user. Cores automatically lock on nonpayment of the bill (i.e., non-receipt of fresh activation code). The model might also allow specification of ceiling and floor levels for chip usage and performance that the user is willing to pay for; the chip could have support for adjusting the core usage based on the user’s current computational requirements.

As opposed to the previous models, the user does not own or rent computational capability in the *PayPerUse* model. Capability is simply treated as a service (for example, like electricity) that is paid for at the end of each billing cycle. The primary advantage of this model over the previous models is that payment is made for the exact usage as opposed to estimated usage. Therefore, overpayments are avoided. This model also has less management overhead than the previous models.

Note that other variants of these models may also consider the diminishing value of hardware over time. I.e., downgrades for refund may become unattractive for the user after some time. Similarly, it may not be profitable for the vendor to provide upgrades after a certain period of time. Discussing the various variants is beyond the scope of this paper. The next section describes the architectural framework required to implement these models.

4 Architectural Framework for the Proposed Models

The framework required to support and enforce the proposed economic models for a many-core chip needs to have an interface at the user side for communicating to the vendor the unique ID of the chip and the number (and possibly IDs) of the cores to be locked/unlocked. The user may also need to communicate chip usage statistics, as well as payment information. The vendor needs to maintain a database indexed by unique chip ID for every chip shipped. This database is used to process payment information as well as to generate appropriate activation codes. Activation codes are used to selectively lock/unlock a subset of cores on the many-core chip. The framework also needs to support secure communication of activation codes and the corresponding configuration of the many-core chip. Configuration involves selective locking/unlocking of cores. The many-core chip also needs mechanisms for license management and verification – this is especially important for models that use activation codes for selective enforcement of a lease. Finally, there needs to be a support on the chip for robust and secure isolation of cores to prevent access to non-licensed cores. This maintains the integrity of the proposed models.

The actual details of the framework depend greatly on the economic model. For example, the mechanism for activating cores in the *UpgradesOnly* model can be implemented with simple one-time activation codes. More support would be needed for the *Limited Up/Downgrades* and *CoresOnRent* models which allow hardware cores to be enabled and disabled many times over the lifetime of the chip. We now discuss the details of the architectural framework for each proposed model.

4.1 UpgradesOnly

The vendor assigns to each chip a unique *Chip_ID* (e.g. a serial number) that is public, i.e., the ID is known to the buyer/user, and a *RUB_ID* that is generated by a physically unclonable function (PUF) ID [32, 26] and is private (i.e. the user does not have access to this ID even though the PUF circuit is present on the chip).

When the user wants to upgrade by X cores, she sends payment information to the vendor along with the Chip_ID to uniquely identify her chip. If the many-core chip is homogeneous, the value of X is also sent. If the many-core chip is heterogeneous, a vector that exactly identifies the cores to be activated is sent.

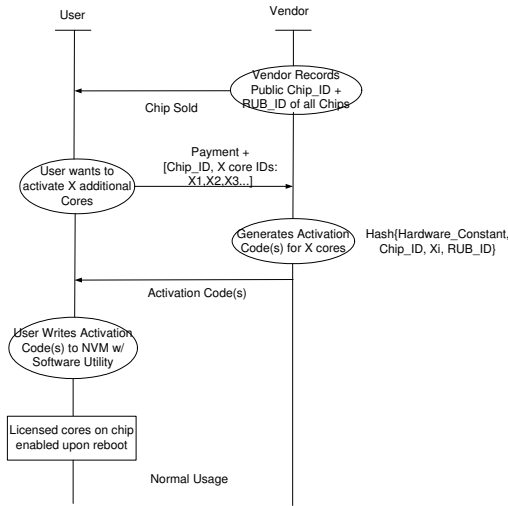


Figure 2. Architectural Framework for *Upgrades-Only* for Heterogeneous Many-cores. IDs of the cores to be activated are included in the secure hash

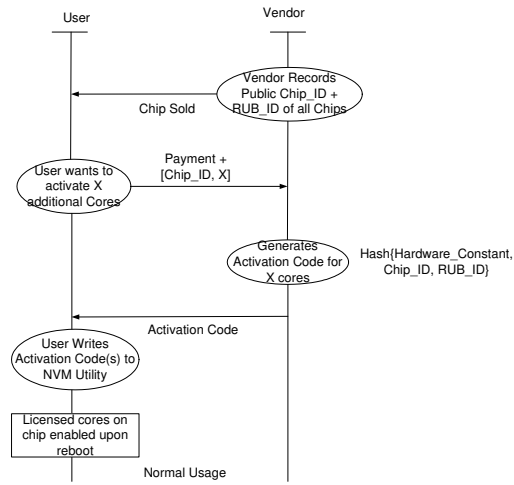


Figure 3. Architectural Framework for *Upgrades-Only* for Homogeneous Many-cores. The number of cores to be activated is included in the secure hash.

The vendor generates activation codes for the X cores by hashing a known common hardware constant for the chip model with the particular Chip_ID, the number of cores (or the ID of a core), and the RUB_ID of the chip. This is equivalent to encryption using a private key in symmetric cryptography [28]. The encrypted codes are then sent to the user that has the private key (implemented using a PUF circuit) and written to a non-volatile memory (that can exist off-chip or on-chip). When the system is rebooted the activation codes are read from the NVM and the cores corresponding to verified codes are then enabled.

Figure 2 and Figure 3 illustrate the framework details for *UpgradesOnly*. Section 5 discusses the architectural details.

4.2 Limited Up/Downgrades

The framework details for *Limited Up/Downgrades* remain identical to those for *UpgradesOnly* for hardware upgrades. However, due to underutilization, etc., the user may at some point decide to deactivate some cores. She first adjusts the number of activated cores by writing into the user-accessible portion of NVM and restarting the system. Restart results in only the new number of cores to be unlocked. The chip also supports a mechanism that places a verification code in NVM on restart, which the user then conveys to the vendor. This code is a proof for the vendor

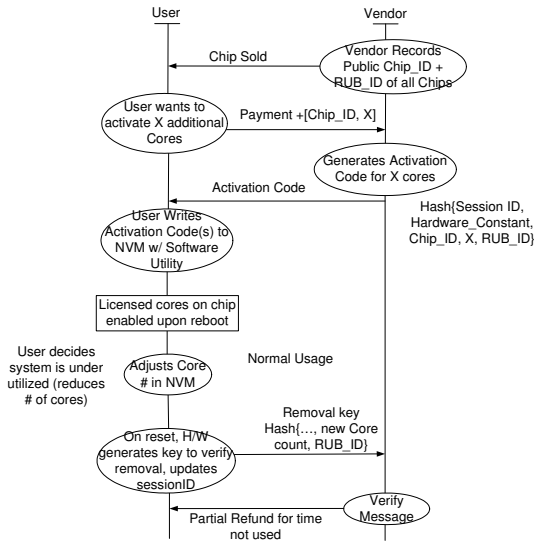


Figure 4. Architectural Framework for *Limited Up/Downgrades*. Hardware generates key when the user downgrades. Vendor verifies key before reimbursement.

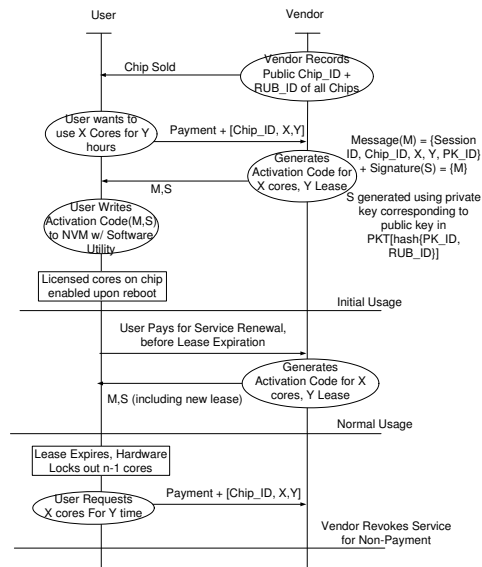


Figure 5. Architectural Framework for *CoresOnRent*. Lease timer locks all cores but one if rent not paid.

that the user’s hardware has indeed been downgraded. After the vendor verifies this information, they can then give the user a (partial) refund for computational capability that the user gave up.

Figure 4 illustrates the framework details for *Limited Up/Downgrades*. Section 5 discusses the architectural details.

4.3 CoresOnRent

In the *CoresOnRent* model, the user provides the vendor the *Chip_ID*, the number of cores that she wants to be activated, and the period over which she wants to use the rented cores. The vendor uses this information to generate activation codes. Also, since the number of possible activation codes used to communicate leases to the hardware is relatively large, a more secure encryption method would be needed. We assume public key cryptography where the activation message is digitally signed. The activation codes are written to the NVM. When the system reboots, the activation codes are verified and then applied to renew the lease (by updating a lease timer - discussed in Section 5).

If the activation codes are not obtained in time, lease expires (the lease timer down counts to zero) and the access to the leased cores is automatically revoked (i.e., the cores get locked). Cores can be unlocked by obtaining new activation codes. These transactions are continually repeated as long as the user wants to access additional hardware on the chip.

Figure 5 illustrates the framework details for *CoresOnRent*. Section 5 discusses the architectural details.

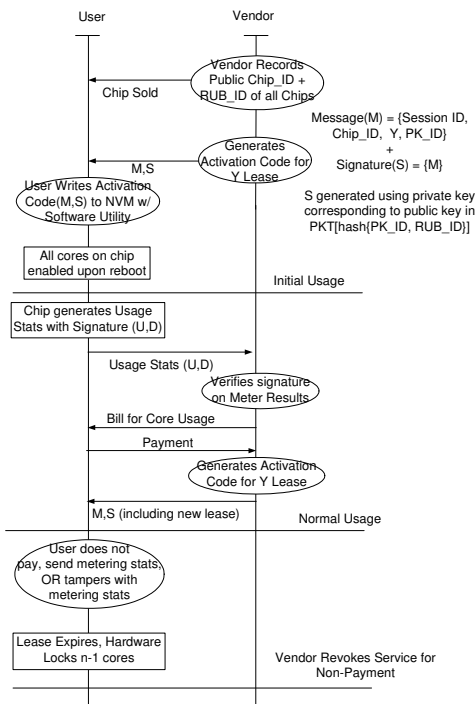


Figure 6. Architectural Framework for *PayPerUse*. Cores are locked if the user does not send usage stats, tampers with stats, or does not pay for service.

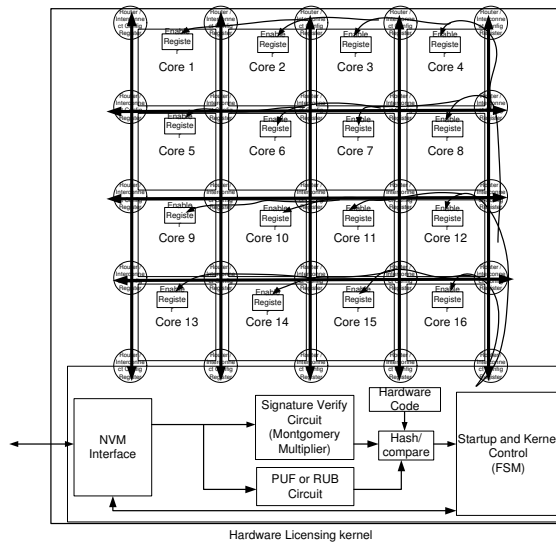


Figure 7. On-chip Hardware Support for License Control. Logic included for interfacing with an NVM, authentication, and isolation control

4.4 PayPerUse

The framework for *PayPerUse* can be built on top of any of the previous models. We describe below a framework built on top of *CoresOnRent*.

The user again must submit a request for a lease to use the hardware. The chip starts with a simple lease for a period of time. The user can use as many cores on the chip as she wants, and usage statistics will be recorded by the chip itself. Periodically, before the lease expires, the chip will amortize the metering results and save it to the memory with a corresponding signature. The user will then send this data back to the vendor. Using the signature to verify the results, the vendor can then produce a bill for the amount of core utilization. When the user pays for the service, the vendor will then produce another activation code and a new lease. The process repeats as long as the user does not tamper with metering results, sends metering results to vendor, and pays for service within billing period. If the vendor determines that any of the conditions for the service contract are not met, the user will not be provided with a new lease, and the chip will automatically lock $n - 1$ cores at the end of the current lease's lifetime.

Figure 6 illustrates the framework details for *PayPerUse*. Section 5 discusses the architectural details.

The next section describes the hardware support required to support the discussed frameworks.

5 Hardware Support

The framework(s) described in the previous section can be implemented using a combination of virtualization-based, firmware-based, and hardware-based techniques. While virtualization and firmware-based techniques are flexible, hardware-based approaches are more secure and prevent tampering. In this section, we discuss hardware-based techniques to implement the discussed framework(s) as we believe that integrity, not flexibility, should be the primary goal of the support mechanisms. Also, as we discuss in Section 6, the proposed techniques have small area and power overhead and can therefore be extended to be flexible as well.

Figure 8 shows the basic schematic for a many-core chip that supports (and enforces) one of the proposed economic models. The chip is augmented with a combinational logic block that is responsible for the various licensing specific functions. This includes verifying codes from vendor for each of the cores on chip. Functionality is also added to verify that the codes can be used with the specific piece of hardware. Additionally, a finite state machine (FSM) block controls the startup sequence, and the various runtime functions (metering, core lockouts from lease expiration, etc.). The chip is initially reset to isolate all cores except $n - 1$ cores, and then activate additional cores which are licensed.

Also included on the chip is an extended register containing a bit vector with an entry for every physical core in the system. This bit vector contains information on which cores are licensed and can be used by the operating system to schedule threads. One of the isolation mechanisms (e.g., power gating, configurable interconnect isolation, routing adjustments in network on chip, etc.) is used to prevent unauthorized access and/or license circumvention. We assume power gating [23] as the mechanism of isolation for this paper.

Activation codes are stored in a non-volatile memory (NVM) module. The NVM contains a portion of secure (not writable by the user) storage and a portion of non-secure (readable/writable by the user) storage. We assume the NVM module to be on-board, off-chip. However, with the recent advances in integration [12, 21], it is not difficult to imagine NVM to be on-chip in near future. Note that activation codes are used to unlock cores, not to lock cores. So, tampering with the NVM does not compromise any of the models. Additionally, the keys and control information from the chip are encrypted with a private key generated by the hardware. So, upon startup, the hardware can determine if the contents of the off-chip storage were tampered with.

Other blocks shown in Figure 8 are described in the following sections.

5.1 UpgradesOnly

Figure 9 and Figure 10 show the data path and control logic needed to support the first economic model.

The on-chip hardware includes a secure NVM controller for accessing both the control data for the licensing manager (non-writable by the user), as well as an insecure portion that is used for storing the user entered activation codes. The on-chip logic also includes a combinatorial hash function (i.e. SHA) for verifying the activation code [18]. Fig-

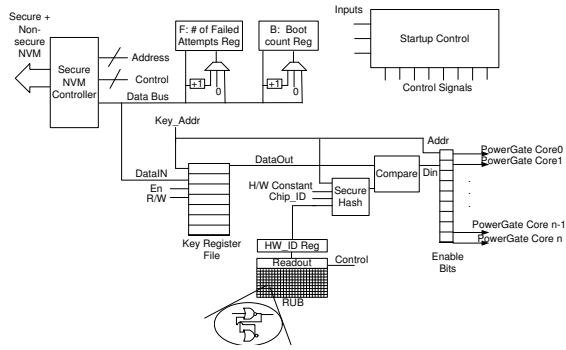


Figure 8. Data path for the *UpgradesOnly* model (heterogeneous many-core)

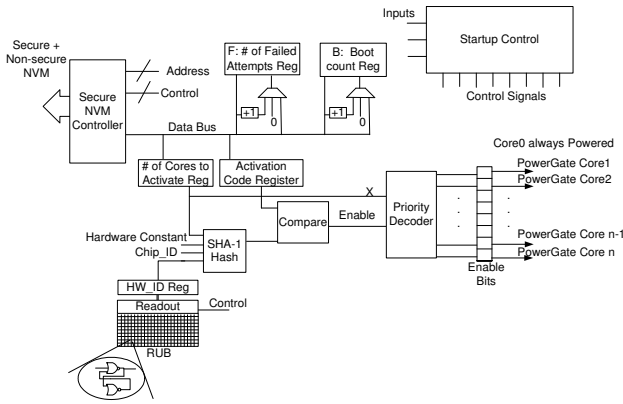


Figure 9. Data path for the *UpgradesOnly* model (homogeneous many-core)

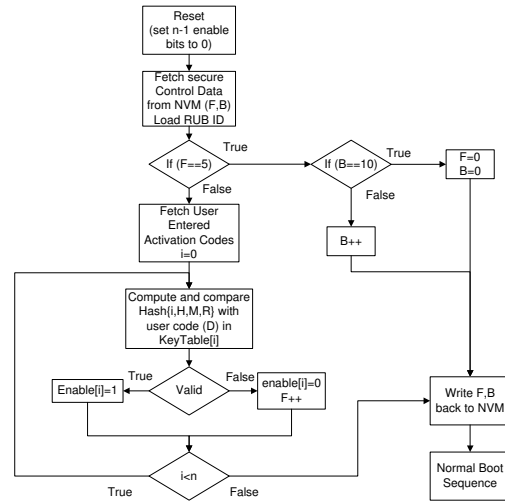


Figure 10. Control Logic for the *UpgradesOnly* model

Figure 8 shows the data path for a heterogeneous many-core processor where activation codes are kept on a per core basis. Figure 9 shows the data path for a homogeneous many-core where a single integer counter is maintained to indicate how many cores should be activated. In either case, a vector of flip flops is used to control the power gating of each core. Thus, no modification is required for any core, beyond routing the enable bits to each power gating signal for each core. In the case of the homogeneous many-core, if the activation code is verified by the hardware, the number of cores will be sent to a priority decoder which selects the proper number of enable bits.

The architecture includes two control registers which maintain the number of failed code verifications and a temporary boot count. A diagram illustrating the control logic is shown in Figure 10. The control registers help mitigate the issue of a user attempting to attack the system using brute force. The chip will not check activation codes for a finite period (i.e. 10 resets) after a certain number of failed activation codes (i.e. 5).

In order to uniquely associate an activation code with a unique piece of hardware, a physically unclonable function (PUF) [32] implementation is also included on the chip. A PUF can be generated by a circuit on the chip which

takes advantage of distinct process variability inherent in the fabrication process. Therefore, chips produced with an identical mask, may generate a unique string of bits for a chip. For example, previous work [17] discusses creating a random unique block(RUB) by using a matrix of cross coupled NOR gates, which amplify the difference in the threshold voltages and produce an output of 1 or 0 at the output. Using the grid of cross coupled nor gates, a unique bit string for that particular piece of hardware can be generated. In all the architectures described below, we assume a similar RUB circuit. We also assume that the vendor has the ability to read the RUB circuit, and maintains a database of Chip_ID, RUB_ID pairs for all the manufactured chips. The user is assumed to not have the ability to read the RUB circuit. Therefore, RUB_ID can be used as the secret piece of information between the hardware and vendor, which can be used for encryption.

The secret and unique hardware id (RUB_ID) can be concatenated on the vendor side with a hardware constant, Chip_ID, and the number of cores to be activated to produce a unique input to the hashing function (SHA). This hash can then be used as the activation code. The hardware on the many-core chip can then verify the code by generating a hash (SHA-1) with the same inputs. When the system is initially powered up, the FSM resets the enable bits, and fetches the secure control data from the NVM. After the code is verified, as described above, the licensing control will write the control data back to the NVM and then continue with the normal bootstrapping process.

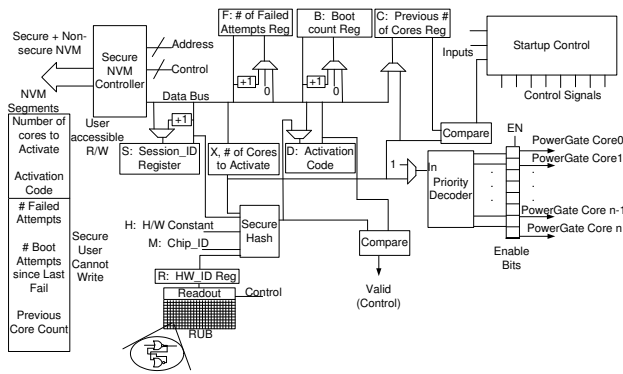


Figure 11. Data path for the *Limited Up/Downgrades* model

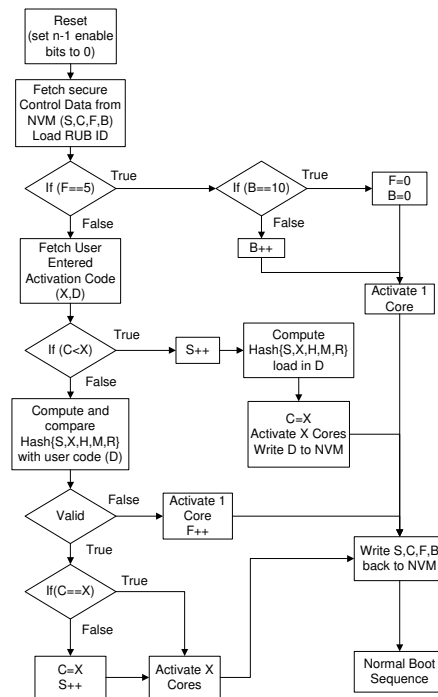


Figure 12. Control Logic for the *Limited Up/Downgrades* model

5.2 Limited Up/Downgrades

An architecture supporting the *Limited Up/Downgrades* model can be easily implemented by making minor modifications to the data path and control logic for the previous architecture. Modifications include a hardware counter that keeps track of the number of times activation codes have been successfully applied, and a register that stores the previous number of active cores. The session counter, known by both the vendor and hardware, would also be included in the hashed value. This would ensure that codes for different system states (upgrades/downgrades) could be differentiated, as different hashes would be produced. The system can then use the control data indicating the last successful number of cores verified to increment the session counter when a different activation code is verified. Additionally, in the case where the user would like to downgrade their system and receive a partial refund, the user first needs to modify the number of active cores in the NVM with the software utility. Upon startup, if the number of cores requested is less than the previous number verified, the hardware can increment the session counter, and produce a hash of the same inputs. The hardware would then write this newly computed hash to the NVM, for the NVM to read and communicate to the vendor. The vendor can then validate this message, and refund the user. Because the session counter is incremented in the hardware, the previous codes used by the vendor, would no longer be applicable. Because a specific core will only be activated/deactivated a limited number of times in the chip's lifetime, this approach is well suited.

Figure 11 and Figure 12 show the data path and control logic for a *Limited Up/Downgrades* architecture.

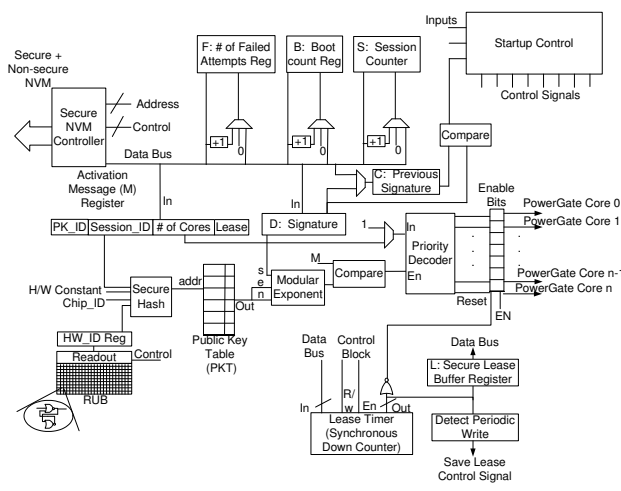


Figure 13. Data path for the *CoresOnRent* model

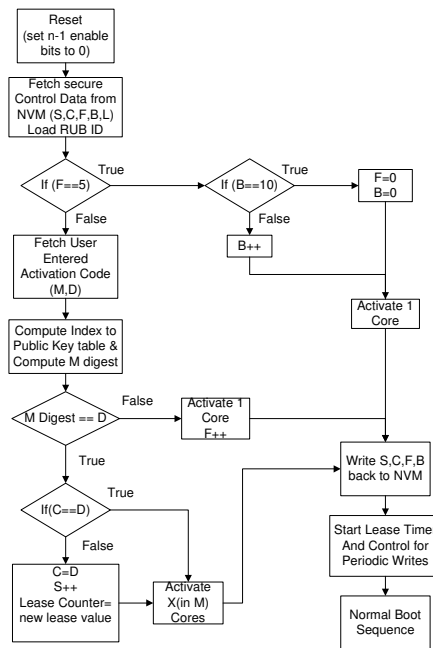


Figure 14. Control logic for the *CoresOnRent* model

5.3 CoresOnRent

The architecture for *CoresOnRent* needs to provide support for implementing a lease. Additionally, there needs to be support for a public key encryption scheme.

To support a public key cryptography system on the chip, we use a hardware-based modular exponentiation [22] technique to verify signatures. The message signed by the vendor with the private key includes the session number, the number of cores to be activated, the lease period, and a number identifying which public key to use for verification. A table with public keys corresponding to the private keys used by the vendor (PKT) can then be used to provide some flexibility in the public/private key pair used by the vendor. A hash of the `Chip_ID`, `RUB_ID`, and hard coded values on the chip can then be used to index into this table. A modular exponent circuit, perhaps implemented with a Montgomery multiplier [33], can then be used on the signature entered by the user. If the result matches the activation message, the control block is signaled for a valid code verification, similar to the previous architectures.

Similarly, we use a hardware-based technique to implement a lease. After enabling the proper number of cores as before, the hardware will compare the newly validated signature to the previously validated one. If they match, control data is written to the NVM, the lease timer is activated, and the normal bootstrap process begins. If the newly validated signature is different, the session counter is incremented, and the new signature and lease value are loaded into their respective timer/register. The new lease value from the activation code is stored in a synchronous down counter (i.e. timer) and activated just before the normal boot process begins. When the timer output equals zero, the reset signal for the enable bits is set high, and all except the first core are power gated. Additionally, circuitry is also connected to the lease timer to detect when to periodically write the timer to the NVM.

Figure 13 and Figure 14 show the data path and control logic for a *CoresOnRent* architecture.

5.4 PayPerUse

A *PayPerUse* architecture shares several of its features with *CoresOnRent* architectures. A key addition to the architecture is the counter added to each core on the chip to record the usage of the cores. In flight instructions are sampled on each core to determine if a core was being utilized by the software stack. If the sampled instruction was a “useful” instruction (i.e. not a NOP), the usage counter is incremented. To prevent compilation techniques from possibly circumventing the metering hardware, the sampling intervals are randomized. The sampling timer is loaded with the output from a random number generator included on the chip. The control block in the licensing manager then periodically cycles through all the usage counters included in the cores, and write these values back to the secure portion of the NVM memory.

The control logic periodically triggers the signing or encryption of the metering results, and writes it to the non-secure portion of the NVM, before the lease expiration. The hardware can use the hashing function for encrypting

the metering results. Alternatively, the architecture can utilize a private key table on the chip, to create a signature of the metering results. In either case, a robust and secure method for the vendor to verify the authenticity of the hardware metering is crucial. The user can then read the metering results from the NVM and send them to the vendor for verification and billing. As long as the user diligently sends metering information, does not tamper with data, and pays for service, the vendor can then generate a new activation message and corresponding signature for the hardware. The new lease in the vendor authenticated message would then be loaded in the control timer on the hardware.

Figure 15 and Figure 16 show the data path and control logic for a *PayPerUse* architecture.

Note that all the architectures described above assume that the user cannot be trusted. In scenarios where the user can be trusted, several of the above functionalities can be offloaded to firmware, middleware, or software.

6 Analysis and Results

To estimate the overhead of providing hardware support for the proposed economic models, we evaluated area, power, and latency overheads for the largest blocks in the hardware licensing kernel. We synthesized both Verilog and VHDL descriptions of these blocks for the UMC 130nm technology using the standard cell library. The Cadence tool-chain was used for hardware description entry, simulation, and synthesis. Cadence NC VHDL and Verilog-XL simulator and the Synopsys frontend for the dc_shell Design Compiler were used for synthesizing the logic and extracting parameters. Area, power, and timing parameters were then extracted for each block. Modelsim was used for initial debugging and concept design. Test benches were created to exercise the main functionality provided by the circuits.

The core functions targeted for our experiments include the authentication and the associated cryptographic logic and a RUB circuit. We considered different encryption methods for verifying messages, including three secure hash(SHA) variants and an RSA crypto system. Simple crypto cores which include sequential logic for the core operations were adapted from open-source cores and created from basic algorithm descriptions. A simple RSA crypto block was created which utilizes a modular multiplication unit repeatedly to verify the public key signatures. Additionally, the secure hash blocks read 32 bits of the activation code off the bus serially and then computes the corresponding hash. One SHA-1 and two SHA-2 (SHA-256 and SHA-512) implementations were also evaluated.

An approximate hardware description for the RUB circuit was also created for two different ID sizes. The basic description involved a matrix of cross coupled NOR gates (SR latches), with an additional fraction of the total number of latches used for dummy circuitry surrounding the matrix of functional latches generating the ID.

Figure 17, Figure 18, and Figure 19 show the area, power, and timing overheads respectively for the various logic blocks. Results show that the overhead for blocks required for hardware supported licensing is minimal compared to the area/power requirements of the pool of computational resources that they manage. For example, the size of the

the authentication block compared to a $100mm^2$ die is less than 0.05%. Similarly, the power consumption for all functional blocks is relatively small ($< 30mW$).

Another way of interpreting the results is that the strength of security (by using bigger keys, for example) can be increased significantly without much absolute overhead. For example, while there is a 275% increase in area and power going from SHA1 to SHA512 implementation and 390% in area on using a wider modular exponentiation block, the absolute overheads continue to be low. Similarly, even when the RSA message signatures have key sizes of 1024 and 1028 (for these key sizes, the typical time required to break the encryption will be greater than the lifetime of the product), the area and power overheads are small.

Finally, low overheads represent an opportunity to incorporate more flexible/exotic schemes for support and enforcement. For example, the *CoresOnRent* and *PayPerUse* models are more vulnerable than the other two models as significantly higher number of activation codes are generated and validated during the processor lifetime. This allows the potential attacker to analyze patterns and reduce the search space for guessing the activation code. Low overheads for the proposed hardware mechanisms entails that there is significant room for expanding the hardware support for license management, with considerably more logic dedicated to more complex hashes and/or larger digital signatures.

Note that while timing results are shown in Figure 19, the delays from verifying licenses and adjusting the system state are isolated to only the system bootup time.

6.1 Limitations and Potential Concerns

While we present the economic models as a way to match available peak computational capability of a chip to the computational requirements of a customer (thereby potentially enabling higher volumes, lower prices, greater market penetration, and bigger consumer base), the proposed models have their share of potential limitations/concerns.

One concern is the licensing model for software for the proposed architectures. Conventional software licensing models tie licensing to a peak computational capability – for example, the enterprise edition of Windows running on k processors vs the home edition that runs on a uniprocessor. The proposed models allow changes in the available peak computational capability of a chip and may, therefore, require unconventional software licensing models. Note, however, that new licensing models are already being developed/used for multi-core architectures as well as for architectures with a hypervisor/VMM running [31] on top of them. Most such models can be used off-the-shelf for proposed architectures as well.

Another potential concern is in terms of *Amdahl's Law for Pricing*. I.e., is the cost of the processor a big enough fraction of the overall system cost that it would justify using new economic models instead of buying a chip with all the cores unlocked? While the cost of the processor is indeed often a small fraction of the overall system cost, it is due to the limited peak computational capability that the processor provides. It may not hold true once the chip's peak

computational capability exceeds a threshold for a given system. The proposed models give the user the option to switch to a higher peak computational capability for the same system.

Another limitation of the proposed models is the number of distinct power/performance points that can be targeted using the proposed architectures. While we present the economic models as a way to target different power/performance points with the same chip, it may not be always possible to use a chip across drastically different market segments. For example, mobile markets expect certain physical footprint as well as certain *performance/watt* range for their chips. Therefore, it may not be feasible to use a chip developed for desktop market segment in the mobile market simply by locking appropriate number of cores to match the peak computational capability of mobile processors.

Another concern is composability of the proposed techniques. Current system architectures are sometimes non-composable and associate the memory system design with a particular number of cores. Changing the number of cores in such cases may result in system imbalance. We believe that memory system design should be composable to support system balance in face of voltage/frequency scaling, clock gating, power gating, core sparing, etc.

Finally, the proposed models involve transactions with the vendor and may raise security and privacy concerns. While we believe that the proposed architectures have adequate support for security and authentication, a user may be hesitant to share her chip information with a third party. Note, however, that a hesitant user always has the option to buy a processor with adequate peak computational capability and not communicate with the vendor regarding upgrades/downgrades.

7 Summary and Conclusions

Current economic models for multi-core processors require a customer to estimate her average-case or worst-case computational requirements beforehand and then buy a processor/system that meets those needs. This causes the computational resources on chips to not match the computational requirements of customers and/or applications, thereby resulting in processors that are often underutilized or over-utilized. Similarly, current models assume one-to-one mapping between a chip and its available peak computational capability. This results in a widening gap between the computational requirements of a customer and the the peak computational capability of the best-fit chip. Both the problems get exacerbated as number of cores on a processor die increase.

This paper presents four economic models for many-core computing. The proposed models provide the customer the flexibility to change the available peak computational capability of the chip (or effective number of cores) during the lifetime of the chip. One of the proposed models also allows the user to pay only for the amount of usage. The models also cover a big spectrum of users with different amounts of variations in their computational requirements.

We show that the proposed models can be supported and enforced easily with a small amount of hardware support. We present hardware techniques to provide this support while honoring security and privacy concerns.

As we enter a world where the number of cores on a die may be large and the profit margins on the hardware low, the economic models that allow peak computational capability of a chip to match the computational requirements of a customer can potentially enable higher volumes, lower prices, and bigger consumer base.

References

- [1] AMD Price List: http://www.amd.com/us-en/corporate/virtualpressroom/0_51_104_609_00.html?redir=cpr01.
- [2] Cellphone Models and Plans: <http://www.easyaccessories.com/cellphone-type.html>.
- [3] Cisco CRS-1 Router: <http://www.cisco.com/en/us/products/ps5763/>.
- [4] ClearPath Mainframe Systems: Pay for use and Metering: http://www.unisys.com/products/mainframes/pay_d_for_d_use.htm.
- [5] From a Few Cores to Many: A Tera-scale Computing Research Overview : http://download.intel.com/research/platform/terascale/terascale_overview
- [6] <http://www.freepatentsonline.com/20050071639.html>.
- [7] <http://www.skmurphy.com/blog/2007/04/11/asic-design-starts-dropping-implications-for-eda/>.
- [8] IBM Capacity on Demand: <http://www-03.ibm.com/systems/p/advantages/cod/index.html>.
- [9] Intel Price List: http://www.intel.com/intel/finance/pricelist/processor_price_list.pdf.
- [10] Intel Xeon Processor 7000 Sequence: http://www.intel.com/products/processor/xeon7000/specifications.htm?iid=products_xeon7000+tab_specs.
- [11] International Technology Roadmap for Semiconductors 2003, <http://public.itrs.net>.
- [12] Kilopass XPM, eXtra Permanent Memory: <http://www.kilopass.com/product.html>.
- [13] Radeon 9500 Pro: <http://ati.amd.com/products/radeon9500/radeon9500pro/index.html>.
- [14] Sharky Processor Pricewatch: <http://www.sharkyextreme.com/guides/wcpg/index.php>.
- [15] SUN Utility Computing: <http://www.sun.com/service/sungrid/index.jsp>.
- [16] N. Aggarwal, P. Ranganathan, N. P. Jouppi, and J. E. Smith. Configurable isolation: Building high availability systems with commodity multi-core processors. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 470–481, New York, NY, USA, 2007. ACM.
- [17] Y. M. Alkabani and F. Koushanfar. Active hardware metering for intellectual property protection and security. In *16th USENIX Security Symposium, 2007*, pages 291–306, 2007.
- [18] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, pages 1–15, London, UK, 1996. Springer-Verlag.
- [19] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer. High-performance CMOS variability in the 65-nm regime and beyond. *IBM J. Res. Dev.*, 50(4/5):433–449, 2006.
- [20] D. C. Bossen, A. Kitamorn, K. Reick, and M. S. Floyd. Fault-tolerant design of the ibm pseries 690 system using power4 processor technology. *IBM Journal of Research and Development*, 46(1):77–86, 2002.
- [21] L. Chang, C. Kuo, H. Chenming, A. Kalnitsky, A. Bergemont, and P. Francis. Non-volatile memory device with true cmos compatibility. *IEEE Electronic Letters*, 35(17):1443–1445, 1999.
- [22] T. Hamano, N. Takagi, S. Yajima, and F. P. Preparata. O(n)-depth modular exponentiation circuit algorithm. *IEEE Trans. Comput.*, 46(6):701–704, 1997.
- [23] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose. Microarchitectural techniques for power gating of execution units. In *ISLPED '04: Proceedings of the 2004 international symposium on Low power electronics and design*, pages 32–37, New York, NY, USA, 2004. ACM.
- [24] F. Koushanfar and G. Qu. Hardware metering. In *DAC '01: Proceedings of the 38th conference on Design automation*, pages 490–493, New York, NY, USA. ACM.
- [25] F. Koushanfar, G. Qu, and M. Potkonjak. Intellectual property metering. In *IHW '01: Proceedings of the 4th International Workshop on Information Hiding*, pages 81–95, London, UK, 2001. Springer-Verlag.
- [26] K. Lofstrom, D. W.R., and D. Taylor. IC Identification Circuit Using Device Mismatch. *Solid-State Circuits Conference, 2000. Digest of Technical Papers*, pages 372–373, 2000.
- [27] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller, and M. Upton. Hyper-threading technology architecture and microarchitecture. *INTEL TECHNICAL JOURNAL*, 6(1):4–15, Feb. 2002.
- [28] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996.
- [29] H. Onodera. Variability : Modeling and Its Impact on Design (Signal Integrity and Variability, Special Section, VLSI Design Technology in the sub-100nm era). *IEICE transactions on electronics*, 89(3):342–348, 2006.
- [30] E. Schuchman and T. N. Vijaykumar. Rescue: A microarchitecture for testability and defect tolerance. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 160–171, Washington, DC, USA, 2005. IEEE Computer Society.
- [31] J. Smith and R. Nair. *Virtual machines: versatile platforms for systems and processes*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2005.

- [32] G. E. Suh and S. Devadas. Physical unclonable functions for device authentication and secret key generation. In *DAC*, pages 9–14. IEEE, 2007.
- [33] A. F. Tenca and C. K. Koc. A scalable architecture for modular multiplication based on montgomery’s algorithm. *IEEE Transactions on Computers*, 52(9):1215–1221, 2003.
- [34] W. Tschudi, T. Xu, D. Sartor, , and J. Stein. High-Performance Data Centers: A Research Roadmap: <http://repositories.cdlib.org/lbnl/lbnl-53483/>. 2004.
- [35] P. M. Wells, K. Chakraborty, and G. S. Sohi. Adapting to intermittent faults in future multicore systems. In *PACT '07: Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques (PACT 2007)*, page 431, Washington, DC, USA, 2007. IEEE Computer Society.
- [36] L. Yuan, P. R. Pari, and G. Qu. Soft IP Protection: Watermarking HDL Codes. In *Information Hiding*, pages 224–238, 2004.

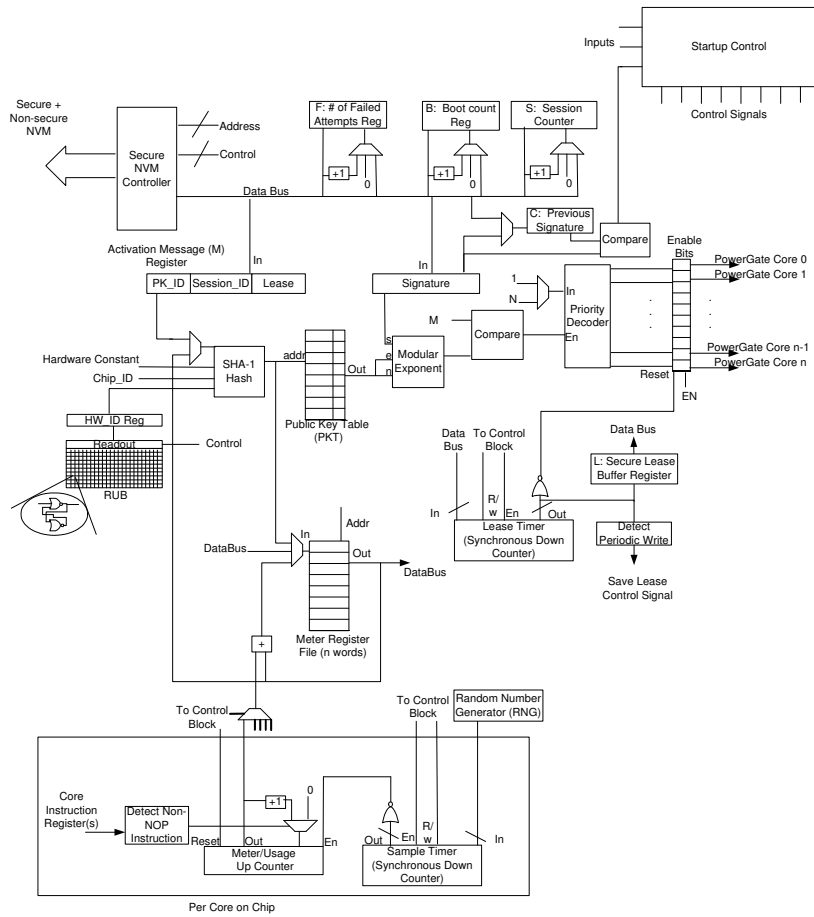


Figure 15. Data path for the *PayPerUse* model

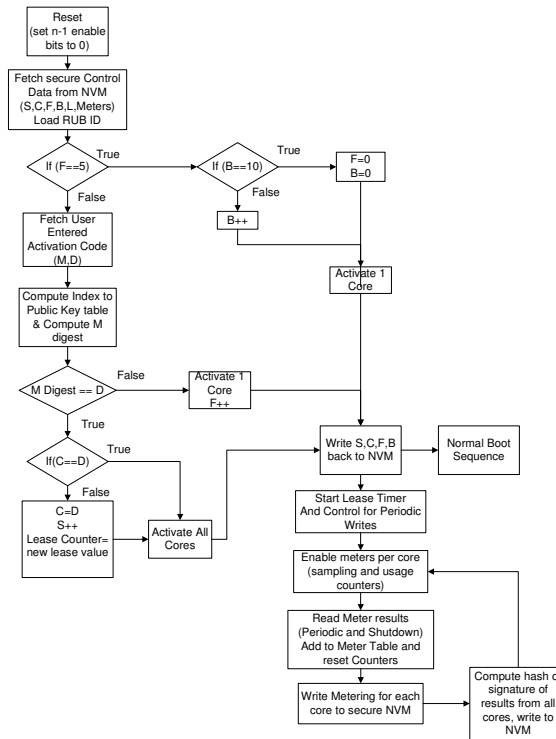


Figure 16. Control logic for the *PayPerUse* model

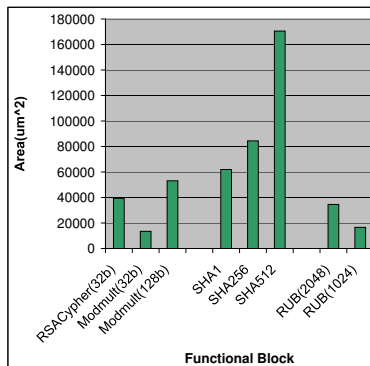


Figure 17. Area Overhead for basic blocks used in hardware licensing kernel

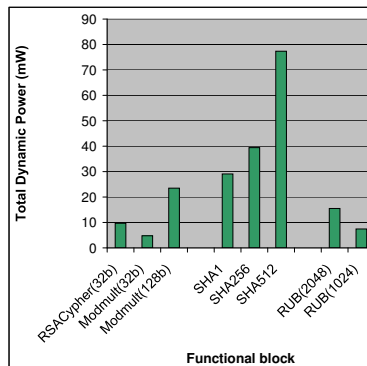


Figure 18. Power Overhead for basic blocks used in the hardware licensing kernel

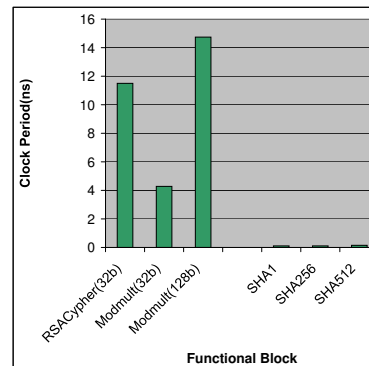


Figure 19. Timing Overhead for basic blocks used in the hardware licensing kernel