System-level Design Space Exploration for Security Processor Prototyping in Analytical Approaches [*]

Yung Chia Lin Chung Wen Huang Jenq Kuen Lee
Department of CS, The National Tsing Hua University, Hsinchu 300, Taiwan
{yclin, cwhuang, jklee}@pllab.cs.nthu.edu.tw

Abstract— The customization of architectures in designing the security processor-based systems typically involves time-consuming simulation and sophisticated analysis in the exploration of design spaces. In this paper, we present an analytical modeling strategy for synoptically exploring of the candidate architectures of security processor-based systems. of We demonstrate examples to employ our analytical models for design space explorations of embedded security systems to deal with scalability issues and architecture constraints. The experiments with the cycle-accurate simulation exhibit the applicability of analytical modeling: average prediction error is less than 10% while speed improvement is in several orders of magnitude.

I. Introduction

As security issues involved in network-aware events such as e-business, network banking, and virtual private networks become more and more important, security mechanisms that require increasing computation capabilities are developed. To deal with the amount of data communication and intensive computation given by security mechanisms, security processors are often needed to provide dedicated security processing to accelerate these processes. Security processor architectures with heterogeneous crypto engines, parallel channel processing, distributed internal bus connection, and memory controllers are interesting for architecture performance evaluations in order to provide design decisions for security processors and systems.

In the case of security processor (**SP**) design, the issue of design space exploration may have various complexities because of the security requisite difference among different applications and appliances. Comparing to those in the case of typical algorithm-specific custom hardware done with cryptography ASIC design, the design space of **SP**-based system is larger and involves a combinatorial aspect in addition to traversing the parameter spaces of the different components. Meanwhile, these **SP**s may also be designed to serve in multiple application scenarios and may be required to support different traffic classes dynamically so that they should be flexible to be able to incorporate new functionality [5]. It is needed to investigate methods that help identify limitations and bottlenecks in system implementation, as we don't want to go all the way down to complete implementations to realize system bottlenecks. However, the growing complexity of the system will result in long simulation time so that exploring design spaces even through high-level architectural simulations would be unfeasible for exploring the huge design spaces. Therefore, there is a need for other methods of performance evaluation such as analytical performance modeling to efficiently opt reasonable design spaces in early design phases. Related research works [2–4, 6] were brought out in recent years as well.

In order to account for all of these issues early at design stage to decrease the overall time-to-market and hardware/software co-simulation efforts, we present an exploration strategy by means of a sufficient general analytical modeling technique for determining a practical design of **SP**s that meet the security requirements and possible trade-off considerations. At the very early design stage, we would like to use high-level analytical modeling for performance estimation to explore a larger part of the design spaces in a limited time period. For some resemblance to the study in [3], we currently use a probabilistic-statistical conception for analytical modeling technique based on the observation of workload characteristics to shorten the gap of analytical modeling estimation and cycle-accurate simulation. Our analytical model is actually used for the design of a family of **SP**s done in the ongoing integrated research project [9–12]. We also demonstrate with examples how to employ our analytical models for design space explorations with embedded security systems to deal with scalability issues and architecture constraints. The experiments with the cycle-accurate simulation exhibit the applicability of analytical modeling that average prediction errors are less than 10% while speed improvement is in several orders of magnitude.

The remainder of this paper is organized as follows. We introduce the security processor in Sec. II. Next, we present the analytical model in Sec. III. The experiments and the discussions are presented in Sec. IV. Finally, the conclusion is then given in Sec. V.

II. Security Processor Architecture

The main feature of our security processor architecture is the scalability of cryptographic functions. To achieve the feature, the internal buses are constructed inside this security proces-

[*]The work was supported in part by NSC-93-2213-E-007-025, NSC-93-2220-E-007-020, NSC-93-2220-E-007-019, MOE research excellent project under grant no. NSC93-2752-E-007-004-PAE, MOEA research project under grant no. 92-EC-17-A-03-S1-0002, no. 93-EC-17-A-03-S1-0002 and an Intel research grant.

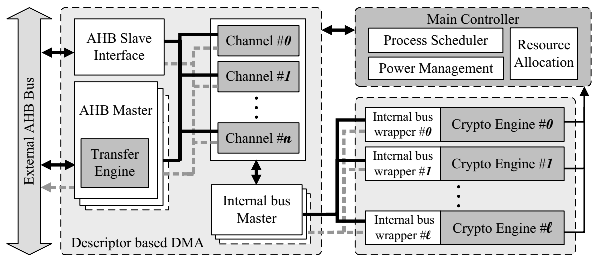

Fig. 1. Security processor architecture

sor. Therefore, versatile crypto engines can be integrated into the **SP** by adopting the compatible bus interface wrappers. The processing flow and cryptographic operations are handled by descriptors to reduce the control signals from the main processor and accompany a descriptor-based DMA that promotes the data movements. The descriptor is a data structure which contains the type of en-/de-cryption functions, the encryption key, the length of data, and pointers that indicate the data addresses. The descriptor also has a pointer to the next descriptor, thus the DMA module could utilize the link list of descriptors to gather data without much overhead.

Fig.1 presents the architecture consisting of a main controller, a DMA module, internal buses, and crypto engines. The main controller has a slave interface of external bus which accepts the control signals and returns the operation feedback via the interrupt port. In the main controller, there is a resource allocation module distributing the resources as the descriptor demands. The process scheduler module and power management module are also added in the main controller for task scheduling and low-power control. The DMA module integrates master interfaces of external bus with the channels and the transfer engines. Each channel stores the header of its processing descriptor. Transfer engines pass the data to dedicated crypto engines via the internal bus. The internal buses are designed to support multiple layers for high speed data transmission. Because the execution time of the crypto engine may be varied, the crypto engine will signal the main controller when the operations are done.

III. Analytical Modeling of Security Processor

The model we developed to use is a simple and appropriate solution for rapid illustration of architectural behavior in the distributed parallel processing design of SOC, which is extended based on super-computer behavior modelings [1, 7, 8]. Assume a generalized design of our target co-processor architecture will incorporate a multi-channel DMA controller, several different types, and amounts of processing-engine backends, which connect each other via several internal buses. Assume incoming requests have been partitioned to run in parallel on each channel of the system, where data processing has been distributed over the processing-engine modules in some static manner determined by the hardware or the software on the host processor. To illustrate the execution of the process , we segment a complete data flow into the following five steps. Once a channel is set up by the DMA descriptor, it will first transfer data from the source memory on the host into the specified channel. It then requests the specified internal bus to forward the incoming data into the designated processing-engine module. When the incoming data are ready for the module, the module will process the data by the request of the descriptor. After the data are processed and ready for outgoing, it will request the specified internal bus to forward the data into the original channel and then transfer the outgoing data to the destination memory on the host.

The execution of every operation can be viewed as a procedure that a channel requests the internal bus twice to serve the data transmission and requests the processing-engine module to serve the data manipulation. Assume each channel execution can be treated as an exponentially distributed random process which produces sets of service request that consists of three correlated operations in the fixed order: two for the internal bus, one for the processing-engine module. We can view each of the k^{th} processing-engine module and the j^{th} internal bus as a server with a constant service rate of M_{s_k} *or* M'_{s_j} bits per second. Let $P_{i,k}$ be the probability that channel i makes its next service request to the processing-engine module k. Define Φ_i to be the average fraction of the time that the i^{th} channel is not waiting for a service request to be completed from any of processing-engine modules and internal buses. In other words, Φ_i is the average fraction of the time that the i^{th} channel spends transmitting data across system bus to/from the host memory. $1 - \Phi_i$ is then the fraction of the time that the i^{th} channel is busy waiting for a service request to complete by any one of processing-engine modules and internal buses. Also, let $\Omega_{k,i}$ and $\Omega'_{j,i}$ be the fraction of the time spent by the i^{th} channel waiting for a service request to processing-engine modules k and internal bus j, respectively.

We would like to determine Φ_i but this depends on the workload of all possible operations caused by programs on the host processor as well as the task scheduling on the target co-processor. In particular, Φ_i is a function of the static density of the security processor service activities in the program on the host processor and the distribution $P_{i,k}$ of those service requests over processing-engine modules. If we consider the activity of one channel, it can be characterized to repeat the following activity pattern; it transfers data from the host memory for a number of cycles, makes a processing-engine module service request, and then transfers data to the host memory for a number of cycles. On the system of processor architecture similar to what mentioned above, it takes time to prepare the encryption or decryption service request and to ship the data over system bus and internal bus into the processing-engine module. When the processing-engine service request arrives at the specified processing-engine module, the request is queued and eventually served by the processing-engine module. The result then returns over buses and is stored into the host memory by the channel which turns back to the available state. Let $system\_cycles_i$ be the total time spent by the i^{th} channel on data transmission over system bus (including host memory accessing, descriptor processing, waiting for memory contention, waiting for system bus contention, etc). We now give the definition for $request\_cycles_i$ and $idle\_cycles_i$. The $request\_cycles_i$ has two elements. It includes the total time spent by the i^{th} channel on preparing processing-engine request and internal bus request, and waiting for processing-engine contention and internal bus contention. Moreover, it includes the overhead time of data traversing among the channel, internal buses, and processing-engine modules, excluding the actual time of data transmission over internal buses. The $idle\_cycles_i$ is the total time of no operation state (no descriptors in descriptor buffers).

Now let

$$channel\_cycles_i = system\_cycles_i + request\_cycles_i + idle\_cycles_i$$

Define

$$M_{r_{k,i}} = \frac{data\_amount_{k,i}}{channel\_cycles_i}$$

which is the fraction of the time that the i^{th} channel spends handling descriptors of processing-engine service requests and preparing data, not including the time of having requests serviced by the k^{th} processing-engine module and the internal bus. If we neglect the interaction between channels and assume that all internal buses are utilizable by all channels and all processing-engine modules, then we have the following modeling.

Model of SP. Let

$$\eta_k = \sum_{i=1}^{n} P_{i,k} \frac{M_{r_{k,i}}}{M_{s_k}}$$
$$\lambda_k = \frac{(1+\varepsilon_k)M_{s_k}}{\sum_{j=1}^{m} M'_{s_j}},$$

where ε_k is the average ratio of the output data size to the input data size in the processing of the k^{th} processing-engine module. The average time that each channel spends doing initiating, host memory communication, and descriptor processing Φ_i is related to the time spent waiting, $\Omega_{k,i}$ *and* $\Omega'_{j,i}$, as the following equations.

$$\Phi_i + \sum_{k=1}^{l} \Omega_{k,i} + \sum_{j=1}^{m} \Omega'_{j,i} = 1 \tag{1}$$

$$\prod_{i=1}^{n}(1-\Omega_{k,i}) + \eta_k \Phi_i = 1 \tag{2}$$

$$\prod_{i=1}^{n}(1-\Omega'_{j,i}) + \sum_{k=1}^{l} \eta_k \lambda_k \Phi_i = 1 \tag{3}$$

Proof. The first equation simply infers that the total fraction of the time which a channel spends busying and waiting on all processing-engine modules and internal buses is exact 1. The second and the third equations are far from self-evident equations like the first one. Let $C_{k,i}$ be the average channel-i-t o-module-k request cycle time for the system, and $\frac{1}{C_{k,i}}$ be the average rate of submitted requests to processing-engine module k for the i^{th} channel. Now let $total\_cycle$ be the total operation time per request, and we get

$$\frac{1}{C_{k,i}} = \frac{data\_amount_{k,i}}{total\_cycles} \tag{4}$$

on average. For a channel, each cycle is either a $channel\_cycle_i$ which is defined earlier or a "request" cycle where the channel is waiting for internal bus service requests or processing-engine service requests being completed. By observing the workloads, we can compute $M_{r_{k,i}}$ which is the ratio of the request data amount to $channel\_cycles$. Based on the definition of $M_{r_{k,i}}$ and equation (4), we can derive

$$\frac{1}{M_{r_{k,i}} C_{k,i}} = \frac{channel\_cycles_i}{total\_cycles} = \Phi_i \tag{5}$$

Moreover, define

$$\delta_{k,i} = \begin{cases} 1 & if\ channel\ i\ is\ not\ waiting\ for\ module\ k \\ 0 & otherwise \end{cases}$$

$$\delta'_{j,i} = \begin{cases} 1 & if\ channel\ i\ is\ not\ waiting\ for\ bus\ j \\ 0 & otherwise \end{cases}$$

Let μ_k be the probability that processing-engine module k is busy and μ'_j be the probability that internal bus j is busy. We have

$$\mu_k = 1 - E(\delta_{k,1}\delta_{k,2}\cdots\delta_{k,n}) \tag{6}$$
$$\mu'_j = 1 - E(\delta'_{j,1}\delta'_{j,2}\cdots\delta'_{j,n}) \tag{7}$$

where $E(\nu)$ is the expected value of the random variable ν. Therefore, $\mu_k M_{s_k}$ and $\mu'_j M'_{s_j}$ are the rate of completed requests to processing-engine module k and internal bus j, respectively. When the system is in equilibrium, $\mu_k M_{s_k}$ is equivalent to the rate of submitted requests to processing-engine module k and $\mu'_j M'_{s_j}$ is equivalent to the rate of submitted requests to internal bus j. Since $\frac{1}{C_{k,i}}$ is the average rate of submitted requests to processing-engine module k from one channel, $\sum_{i=1}^{n} \frac{P_{i,k}}{C_{k,i}}$ is the total rate of submitted requests to processing-engine module k from all channels. Consequently, we have the equivalence,

$$\sum_{i=1}^{n} \frac{P_{i,k}}{C_{k,i}} = \mu_k M_{s_k} \tag{8}$$

Likewise, $\sum_{k=1}^{l}\sum_{i=1}^{n} \frac{P_{i,k}}{C_{k,i}}$ is the average rate of submitted requests to all internal buses from all channels. However, submitted requests to internal buses also include requests from all processing-engine modules whereas the processed data are transferred back to channels when processing-engine modules services are completed. Due to the law of data indestructibility, we can have $\sum_{k=1}^{l}\sum_{i=1}^{n} \frac{P_{i,k}(1+\varepsilon_k)}{C_{k,i}}$ to be the average rate of submitted requests to all internal buses from all channels and all processing-engine modules. Accordingly, we have the equivalence as follows:

$$\sum_{k=1}^{l}\sum_{i=1}^{n} \frac{P_{i,k}(1+\varepsilon_k)}{C_{k,i}} = \sum_{j=1}^{m} \mu'_j M'_{s_j} \tag{9}$$

By combing equations (5), (6), (7), (8), and (9), we get

$$\begin{cases} \eta_k = \sum_{i=1}^{n} P_{i,k} \frac{M_{r_{k,i}}}{M_{s_k}} \\ E(\delta_{k,1}\delta_{k,2}\cdots\delta_{k,n}) + \eta_k \Phi_i = 1 \end{cases}$$
$$\begin{cases} \lambda_k = \frac{(1+\varepsilon_k)M_{s_k}}{\sum_{j=1}^{m} M'_{s_j}} \\ E(\delta'_{j,1}\delta'_{j,2}\cdots\delta'_{j,n}) + \sum_{k=1}^{l} \eta_k \lambda_k \Phi_i = 1 \end{cases}$$

Nevertheless, since both $\delta_{k,i}$ and $\delta'_{j,i}$ are binaries, we have by symmetry

$$E(\delta_{k,i}) = 1 - \Omega_{k,i}$$
$$E(\delta'_{j,i}) = 1 - \Omega'_{j,i}$$

for every channel i. We now make a critical approximation by assuming that all the channels have non-correlated activities and we get

$$E(\delta_{k,1}\delta_{k,2}\cdots\delta_{k,n}) = E(\delta_{k,1})E(\delta_{k,2})\cdots E(\delta_{k,n}) = \prod_{i=1}^{n}(1-\Omega_{k,i})$$

$$E(\delta'_{j,1}\delta'_{j,2}\cdots\delta'_{j,n}) = E(\delta'_{j,1})E(\delta'_{j,2})\cdots E(\delta'_{j,n}) = \prod_{i=1}^{n}(1-\Omega'_{j,i})$$

The result follows. □

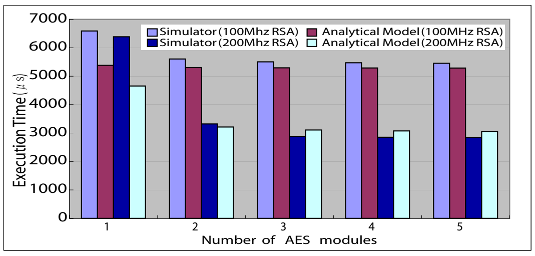

Fig. 2. Comparison between simulator and analytical model

IV. Experiments and Discussions

In the first experiment, it shows the comparisons between the analytical model and the cycle accurate simulator which constructed in SystemC. We condense SSH activities on a real server to emulate higher workloads in 1000 microseconds and to generate input patterns for the **SP** simulator and average input workloads for analytical models. The working frequency of **SP** and 32-bit system bus is simulated at 133MHz. Fig. 2 shows the execution results for a configuration in 1 internal bus, 10 channels, 2 RSA modules, and several AES modules. The number of AES modules is increased from 1 to 5. The internal bus has the same bandwidth with the system bus; one 100MHz RSA module has around 3.17Mbps processing rate; one 200MHz RSA module has 6.34Mbps processing rate; one 100MHz AES module has 706Mbps processing rate. By profiling the operations of a SSH server, The used workload has 3.2Gbps AES encryption data request rate and the RSA demands are around 1% of the AES demands. This diagram illustrates that the execution time is reduced by the increase of AES modules for the particular workload above. The bottleneck for the system is then at RSA, since the increase of AES modules no longer speeds up the performance when the number of AES is over 3. In this experiment, the average prediction error is under 8%. A major portion of the error results from that the simulator assumes no data buffer in the channel, while the analytical model in this experiment assumes the overlapping effects of loading data into channels. It is significant for small numbers of AES modules, as it gives higher inaccuracy with one AES module. The estimation is more accurate when the number of AES module grows. Our analytical model is much faster in giving performance estimations than the cycle-accurate simulator. The analytical model is done by numerical computations through equations (1), (2), and (3), so that the execution time for performance estimation is fixed, but the simulator will depend on the total duration of the workload. In this particular experiment, the toolkit based on our proposed analytical models is over 10000 times faster than the simulator.

Next, we show that our analytical models can be used for performance evaluations and design space explorations. We have built a toolkit based on our analytical model for our **SP** architecture template to rapidly estimate performance behaviors. It integrates the simulator and the analytical evaluator into the unified design space exploration interface as shown in Fig. 3. The statistic workloads contain the real DMA-descriptors with data, which could be gathered by measures of real applications or generated by our developed task pattern generator. The analytical modeling parameter generator can derive the required parameters ($P_{i,k}$, $M_{r_{k,i}}$, ...) of the model introduced in Sec. III and then provide these values to the analytical model evaluator. The evaluation data analyzer finally uses statistic workloads, architecture configurations, and the data from either the analytical evaluator or the simulator to analyze and output the visual analysis results like Fig. 4. These charts would help us to balance the system design for overall considerations such as performance, cost, and power consumption. The toolkit lets us perform hierarchical performance evaluation to save time in the exploration through both the coarse-grained analytical evaluation and the find-grained simulation: we first use the analytical evaluation to reduce the large design space, and then use the simulator to acquire the accurate evaluation in the focused design space. Fig. 4 shows the performance information produced by our **SP** analytical toolkit software in the following experiment. The measured workload has average 3.084Gbps input requests within 10 minutes. Among the input requests, around 0.95% of them are RSA requests. The X-axis gives the possible system configurations. It begins with a parameter set of six channels, one AES engines, and five RSA engines with initial engine speed at 100MHz. It then increases one channel and adds 50MHz to the engine speed for next possible configuration. The time fraction chart gives the dominating elements in the processing time; the processing time chart gives the total processing time; the performance chart gives effective processing rate; the utilization chart gives average utilization for each component. Hardware designers benefit a lot from these performance figures. For example, one can see processing rate converges at 300MHz in the processing time chart. With utilization chart, we can judge the convergence is due to internal bus. Then, we can see the peak performance for the system throughput is 1.5Gbps from the performance chart.

Finally, we use our analytical modeling technique to predict the behavior of different configurations of a security processor that has AES, 3DES, RSA, and MD5-HMAC engines. For exploring the capabilities and adaptability of different designs, we analyze the effects of different workload on these configurations. This exploration is particular important for such scalable designs as our **SP** architecture because we would like to know the correct way of fine-tune hardware configurations to meet application specific requirements. In this experiment, the parameters of AES and RSA engines are the same as mentioned earlier. The parameters of MD5 and 3DES engines are set to deliver nearly 8Mbps and 5.328Mbps per MHz, respectively. Fig. 5 illustrates the performance prediction of several configurations. The "base" configuration is set as three AES engines, two 3DES engines, two RSA engines, one MD5-HMAC engine, one internal bus, and one DMA with six channels and all of them operate at 66MHz. We have additional three configurations to predict: the speed of crypto engines from the "base" configuration has doubled; the number of crypto engine and internal bus from the "base" configura-

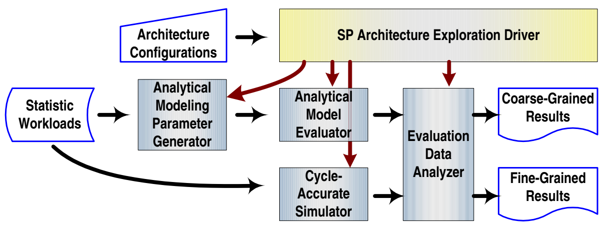
Architecture Configurations
SP Architecture Exploration Driver
Statistic Workloads
Analytical Modeling Parameter Generator
Analytical Model Evaluator
Evaluation Data Analyzer
Coarse-Grained Results
Cycle-Accurate Simulator
Fine-Grained Results

Fig. 3. The integrated analytical toolkits

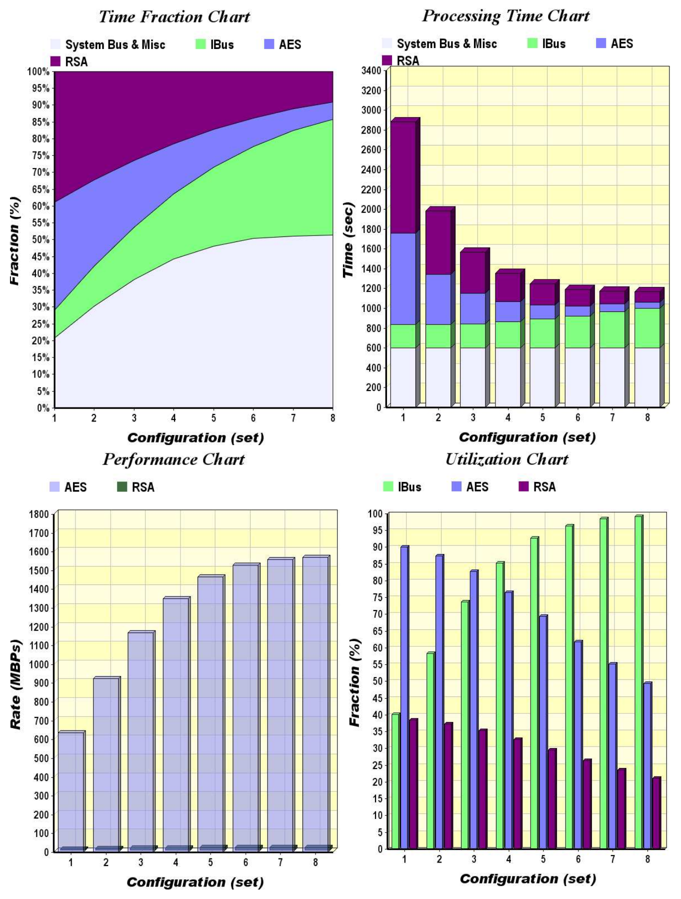

Fig. 4. Visual results of performance evaluation output by the toolkits

tion has doubled; the number of DMA channel, crypto engine, and internal bus from the ”base” configuration has doubled. The initial workload of 793.6 Mbps input rate is distributed as 512Mbps AES requests, 256Mbps 3DES requests, 5.12Mbps RSA requests, and 20.48Mbps MD5-HMAC requests. The X-axis represents various workload sets with the increasing input rate from the original value of the initial workload to 7 times of the original value. The result shows the **SP** with the ”base” configuration serves 340Mbps at initial input rate. The peak throughput is around 465Mbps at 3 times of the initial rate, and down to 442Mbps when the input rate grows. This is due to that the request patterns are mixed with different crypto operations. The delay time among the same type of requests is reduced with the growth of input rate so that the utilization of crypto engines are improved. While the input rate keeps growing, the contention of crypto engines occurs if crypto engines are fully utilized. Fig. 5 shows that the throughput drops down after attaching the peak value of each configuration. Observing the result noted by the ”round” symbol, the throughput of this configuration is smaller than the throughput of the configuration whose result noted by the ”square” symbol at the lower input rate. That is due to the resource contention of internal buses and data sources, which increases when crypto engines have doubled. In the configuration noted by the ”square” symbol, however, the throughput stops improving when the input rate exceeds the limitation of transmission capabilities of internal buses and channels. Thus, there is an intersection of the ”square” line and the ”round” line. This analysis can help us reduce the design space and refine our design on the requisite workload considerations.

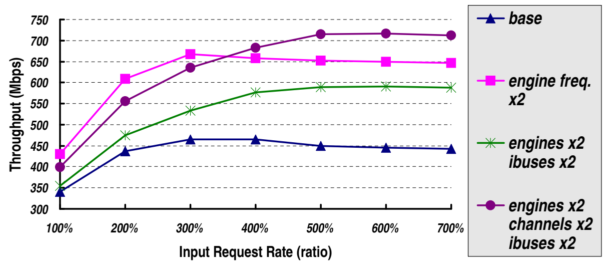

Fig. 5. Performance prediction

V. Conclusion

The analytical modeling techniques provide insights into the performance of system design, reveal bottlenecks, and indicate where tuning efforts would be the most effective without time-consuming simulation. Also, the correlated analysis assists in rapidly exploring the design space for the future systems. In this paper, we have described how to apply the analytical modeling technique developed in this work to our security processor-based system design. The experiments with the cycle-accurate simulation exhibit the applicability of analytical modeling that average prediction errors are about 10% and speed improvement is enormous. We will continue making efforts to improve our analytical models and apply them to more applications such as task scheduling and low power studies in the future.

References

[1] Hwang K and Briggs F. Computer Architecture and Parallel Processing. Mc Graw-Hill, 1984.

[2] Baghdadi A, Zergainoh N.-E, Cesario W.O and Jerraya A.A. Combining a Performance Estimation Methodology with a Hardware/Software Codesign Flow Supporting Multiprocessor Systems. IEEE Transactions on Software Engineering 2002;28;822–831.

[3] Eeckhout L and De Bosschere K. Hybrid analytical-statistical modeling for efficiently exploring architecture and workload design spaces. Proceedings of PACT 2001;25–34.

[4] Kerbyson D.J, Wasserman H.J and Hoisie A. Exploring advanced architectures using performance prediction. Proceedings of International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems 2002;27–37.

[5] Ravi S, Raghunathan A, Potlapally N and Sankaradass M. System design methodologies for a wireless security processing platform. Proceedings of the 39th Design Automation Conference 2002;777–782.

[6] Xinping Zhu and Malik S. A hierarchical modeling framework for on-chip communication architectures. Proceedings of IEEE/ACM International Conference on Computer Aided Design 2002; 663–670.

[7] Daya Atapattu and Dennis Gannon. Building analytical models into an interactive performance prediction tool. Proceedings of ACM Supercomputing 1989;512–530.

[8] Francois Bodin, Daniel Windheiser, William Jalby, Daya Atapattu, Mannho Lee and Dennis Gannon. Performance evaluation and prediction for parallel algorithms on the BBN GP1000. Proceedings of the 4th ACM international conference on Supercomputing 1990; 401–403.

[9] J.-H. Hong and C.-W. Wu. Cellular array modular multiplier for the RSA public-key cryptosystem based on modified Booth’s algorithm. IEEE Transactions on VLSI Systems 2003;11;474–484.

[10] C.-P. Su, T.-F. Lin, C.-T. Huang and C.-W. Wu. A highly efficient AES cipher chip. ASP-DAC 2003;561–562.

[11] M.-C. Lee, J.-R. Huang, C.-P. Su, T.-Y. Chang, C.-T. Huang and C.-W. Wu. A true random generator design. 13th VLSI Design/CAD Symp. 2002;137–140.

[12] M.-Y. Wang, C.-P. Su, C.-T. Huang and C.-W. Wu. An HMAC processor with integrated SHA-1 and MD5 algorithms. ASP-DAC 2004.