20

Enabling Network Security in HPC Systems Using Heterogeneous CMPs

Ozcan Ozturk
Bilkent University, Ankara, Turkey

Suleyman Tosun
Ankara University, Ankara, Turkey

As technology scales, the International Technology Roadmap for Semiconductors projects that the number of cores will drastically increase to satisfy the performance requirements of future applications. Performance improvements brought by multi-core architectures have already been used in network security processors either using homogeneous chip multiprocessors (CMP) or through custom system-on-a-chip (SoC) designs. However, homogeneous CMPs provide only one type of core to match various application requirements, thereby not fully utilizing the available chip area and power budget. On the other hand, the heterogeneous CMP is a better option for a network security processor with programming needs ranging from encryption/decryption to content processing, as it allows the processor to better match the execution resources of application needs. This chapter explores the possibility of using heterogeneous CMPs for network and system security. More specifically, we propose an integer linear programming (ILP)-based methodology to mathematically analyze and provide heterogeneous CMP architectures and task

High-Performance Computing on Complex Environments, First Edition.
Edited by Emmanuel Jeannot and Julius Žilinskas.

distributions that can reduce the energy consumption of the system. Our results indicate that the proposed approach generates better results when compared to a homogeneous counterpart.

20.1 INTRODUCTION

Network security has become a key problem with the increase in the number of Internet users and network traffic. Possible Internet threats and attacks include (i) viruses and worms that infect a computer, (ii) distributed denial of service (DDoS) attacks such as a UDP (user datagram protocol) storm, (iii) trojans that aim data destruction, remote access, or security software disable, (iv) spyware to take partial control of the computer, and (v) spam in many forms such as spam e-mail, chat spam, or blog spam. To overcome these security threats, network security has become increasingly complex with the diversity and number of different security applications. Applications widely used for security include antivirus software, spyware systems, firewalls, intrusion detection systems, antispam systems, and DDoS systems. Network security systems provide more complex security features while sustaining the required throughput for the increasing network bandwidth.

Ideal network security can be provided by real-time data packet inspection using the aforementioned security system tools. The performance of the underlying architecture is crucial for accomplishing such real-time data packet inspection. Traditional single-processor network systems are not sufficient to fully inspect the packets with the increased number of inspection demands and the density of the network traffic. Recently, CMP architectures have been proposed to address the processing requirements of network security systems because CMP technology offers higher performance, scalability, and energy efficiency.

CMPs become an increasingly attractive option for obtaining high performance and low power consumption since it has become an increasingly difficult task to obtain more performance out of single-processor designs. As a result, CMPs are widely available in the market and have been used as an attractive option for overcoming the barriers in processor design. As technology scales, the International Technology Roadmap for Semiconductors projects that the number of cores will drastically increase to satisfy the performance requirements of future applications [1].

One of the most important benefits of a CMP over a traditional single-processor design is the power consumption reduction by reduced clock frequency. Other benefits of CMPs include (i) scalability provided using many dimensions of parallelism such as thread-level parallelism, loop-level parallelism, and instruction-level parallelism; (ii) cost of design and ease of verification, which in turn reduces the time to market and lowers the chip costs; (iii) better area utilization of the available silicon as the cores share common logic; and (iv) faster and cheaper on-chip communication.

In the network security domain, a CMP will be able to provide many advantages over a traditional single-processor design because there are multiple tasks. If the network security system is not fast enough, either packet transmission will slow down or packet inspection will be performed partially. For example, consider a network security processor that ensures security features in real time while processing information. In such a system, the network security processor will need to perform multiple tasks such as firewall, virtual private network (VPN), internet protocol (IP) security, content processing, and cryptography simultaneously [2, 3]. A single processor will not be powerful enough, whereas a CMP-based network security processor will process the packets fast enough while providing the security features, which requires parallel processing.

There are many open issues that need to be addressed in the context of network security CMPs. For example, data mapping, communication optimization, and task partitioning play important roles in a general-purpose CMP design which also applies to network security processors.

- *Data Mapping:* On-chip memory area should be managed carefully because accessing off-chip memory presents large performance penalties even if a small fraction of memory references go off-chip. Moreover, frequent off-chip memory accesses can increase the overall power consumption dramatically.
- *Communication Optimization:* Communication between the cores in the CMP should be performed carefully because the bandwidth between the CMP nodes is limited. Careful data placement and effective communication protocol is key to reducing the communication overheads.
- *Task Partitioning:* Ensuring security while processing the content of the packets requires effective task partitioning. Task partitioning is a well-known problem for multiple processor environments, but it needs a fresh look for network security CMPs as there are additional constraints/requirements and opportunities. For example, certain security-related tasks can be performed on different packets simultaneously, providing parallel processing. On the other hand, one needs to be careful in distributing the security tasks because this may cause a lot of interprocessor data communication.

Performance improvements brought by CMP architectures have already been realized in network security processors either using homogeneous CMPs or through custom system-on-a-chip (SoC) designs. However, homogeneous CMPs provide only one type of core to match various application requirements, thereby not fully utilizing the available chip area and power budget. On the other hand, heterogeneous CMP is a better option for a network security processor with programming needs ranging from firewall and encryption/decryption to content processing, as it allows the processor to better match execution resources of application needs.

Security tasks require a variety of resources such as computational power, memory space, and bandwidth. Compared to a packet inspection unit, a cryptography engine requires more processing power, whereas the former requires more memory space and bandwidth compared to the latter. General-purpose CMPs provide the same type of processor for various security applications. This mapping allocates unnecessary resources to certain types of applications, which increases the power consumption. On the contrary, network security tasks can be mapped onto the appropriate processor in the heterogeneous CMP to meet the performance requirement while minimizing power consumption and development complexity. To strike the right balance, one should be careful in choosing the type of processors and memory components.

This chapter explores the possibility of using heterogeneous CMPs for network and system security. We compare heterogeneous CMPs with homogeneous counterparts and provide experimental evaluation of using both on network security systems. The remainder of this chapter is structured as follows: Section 20.2 explains the related work on CMPs and their use in network security. The details of heterogeneous NoC (network-on-chip)-based CMP architecture and an overview of our approach are given in Section 20.3. Section 20.4 discusses the heterogeneous CMP-based network security processor design and advantages. An experimental evaluation is presented in Section 20.5. Conclusions are provided in Section 20.6.

20.2 RELATED WORK

We present the related work in two parts. First, we summarize the related work on heterogeneous processors in general and their benefits. Second, we explore the related studies on CMP network security processors.

Heterogeneous CMPs have been introduced to provide a wide variety of processing resources to effectively use the available chip area while ensuring enough processing power. Benefits provided by heterogeneous CMPs are discussed from many angles in [4, 5]. In [6], the authors present a model-based exploration method to support the design flow of heterogeneous CMPs. They implement cost models for the design space exploration using several cost parameters such as performance and throughput. The work presented in [7] explores the effects of heterogeneity on commercial applications using a hardware prototype.

On the software side, OpenMP directives are extended [8] to address the heterogeneity in CMPs. Several optimization techniques are extended to utilize advanced architecture features of the target system. In [9], the authors present a multithreaded code generation method that tries to reduce the number of interprocessor communications. They primarily try to reduce the number of messages exchanged, thereby reducing the total communication between the cores. In [10] a method is proposed to parallelize the JPEG compression algorithm on a heterogeneous CMP. Task partitioning with replication is applied on heterogeneous CMPs in [11]. Specifically, the authors employ replication up to a certain threshold in order to improve system reliability. In [12], the authors explore the power–performance efficiency of hyper-threaded heterogeneous CMP servers, and propose a scheduling algorithm to reduce the overall power consumption of a server while not affecting the performance. Becchi and Crowley [13] explore the benefits of heterogeneous CMPs using a dynamic assignment policy that assigns the threads between the cores. One of the early heterogeneous CMP architectures proposed [14] chooses the appropriate core for a task and reduces the energy–delay product by doing so.

From a hardware perspective, [15] explores the processor design problem for a heterogeneous CMP from scratch, as processors designed for homogeneous architectures do not sufficiently map to the heterogeneous domain. They study the effects of processor design in terms of area or power efficiency. A compiler-driven tightly coupled VLIW (very long instruction word) processor is presented in [16]. It is a superscalar processor on a single chip, to improve the performance of single threaded applications while supporting multithreaded applications. More specifically, they use a high-performance VLIW core to run high ILP applications, whereas a superscalar core is used to exploit multithreaded applications. In [17] signal processing architectures at the system level are employed for heterogeneous CMPs, while [18] present a heterogeneous CMP with same instruction set architectures (ISAs) working on different voltage levels and frequencies.

Network security processors have been widely studied in the context of security coprocessor implementations [19–21]. In [19], the authors present a security processor to accelerate cryptographic processing such as RSA, Advanced Encryption Standard (AES), and random number generation. CryptoManiac [20] presents a coprocessor for cryptographic workloads. On the other hand, a generic network security processor for security-related protocols is presented in [21]. With the emerging CMP architectures, CMP security processors have also been introduced. Commercial products to provide network security using CMP architectures are already in the market. Octeon [2] introduced by Cavium has multiple MIPS64 cores, which is used for traffic control, content, and security processing. Similarly, SonicWALL [22] has been proposed to provide a faster packet processing platform through data packet inspection on a CMP architecture. Endian Firewall Macro X2 [3] provides network protection and content inspection with Intel CMP architecture. CMP architectures are used for other security systems [23]. A parallel intrusion detection system by using CMP architecture has been proposed. In [24], the authors present the performance improvement in DDoS defense by using CMP architectures. Deep packet inspection using parallel Bloom filters is discussed in [25].

There have been prior attempts to use task allocation on a network processor using pipelining [26–28]. Shoumeng *et al.* [26] propose a genetic algorithm-based task assignment for network processors. They assign different packet processing tasks to the network processors in a pipelined manner. Similarly, a task allocation scheme for CMP architectures is proposed in [27], where the selective replication of some modules is made to increase the number of tasks running parallel. The authors in [28] introduce GreedyPipe, a heuristic-based task scheduling algorithm on CMP-based network processors in a pipelined manner. A randomized rounding (RR) based solution for task mapping and scheduling is presented in [29]. Our approach is different from these prior efforts, as we try to optimize the energy consumption of a network processor using a heterogeneous CMP architecture.

20.3 OVERVIEW OF OUR APPROACH

20.3.1 Heterogeneous CMP Architecture

Despite the many advantages of CMPs over uniprocessor architectures, one of the key questions raised by many researchers is the effectiveness of CMPs [4, 5]. One aspect of this problem is due to the infancy of the software solutions targeting such architectures. Current programs, compilers, and software architecture techniques, in general, rely on the fact that there is only one core running on the background [30]. Hence, it becomes very difficult to effectively use the underlying processing power. There are some initial attempts to target this problem, but these techniques are still in their infancy [30].

On the other hand, from a hardware point of view, the homogeneity inherent to CMPs is another source of limitation in extracting the best utilization from these architectures. To overcome the limitations due to the homogeneous behavior of CMPs, heterogeneous (asymmetric) CMPs have been proposed [31, 32]. For example, IBM's Cell Processor is a heterogeneous CMP composed of one PPU (power processing unit) and eight SPUs (synergistic processing unit) [31, 32]. In this architecture, PPU is used as the main coordinator, whereas SPUs perform SIMD (single instruction multiple data) processing on mass data.

Even though complex cores provide higher single-thread performance, they consume more area and power compared to their simpler counterparts. Every application has a different processing need and a memory requirement. Even one single application has different requirements throughout its execution. A network security application may exploit a high level of ILP where a powerful core will be a better match, whereas a simpler core will suffice for a different application with lower ILP. Choosing the best match for an application will reduce the power consumption. However, homogeneous CMPs provide only one type of core to match all these various requirements. The ability to dynamically switch between different cores and power down unused cores is the key in heterogeneous CMPs. It allows the processor to better match execution resources with application needs, which in turn enables different workloads from high to low. It was shown that a representative heterogeneous processor using two core types achieves as much as 63% performance improvement over an equivalent-area homogeneous processor [4, 5]. Hence, heterogeneous multiprocessors achieve better coverage of a spectrum of load levels.

In our proposed approach, NoC-based heterogeneous CMP architecture is exposed to the ILP solver. We assume that a heterogeneous NoC-based CMP is a two-dimensional mesh topology, where each node of this mesh consists of a network switch/router, a processor, and a memory hierarchy.

20.3.2 Network Security Application Behavior

Network security applications have different resource requirements such as computational power, memory space, or bandwidth. For example, in a network security system, a packet inspection unit will require more memory space and higher bandwidth compared to a cryptography engine. On the other hand, a cryptography engine will require more processing power to perform multiple encryption/decryption tasks. Previously proposed CMP architectures use general-purpose CMPs which provide the same type of processor for a wide spectrum of security applications. This mapping allocates unnecessary resources to certain applications, thereby increasing the overall power consumption. On the contrary, network security tasks can be mapped onto the appropriate processor in the heterogeneous CMP to meet the performance requirement while minimizing power consumption and development complexity.

Consider a network processor that has two main components, content processing and security processing. The content processing unit performs packet inspection which manipulates packets coming from the physical layer. This process involves multiple tasks including data compression, classification, header modification, and framing. Similarly, security processing involves encryption, decryption, user authentication, and key management. Individual tasks can be mapped onto cores of a given CMP architecture with various performance values. Table 20.1 gives the performance values of different components of a security unit with different processors [19, 20]. This table is obtained using AES encryption/decryption with a 128-bit key, and RSA engine with 1024 bits and HMAC using SHA1. The reported results are within acceptable ranges, where the overall power consumption at 83 MHz is reported to be 383.5 mW. As can be seen from this table, processor requirements may vary depending on the underlying functionality.

20.3.3 High-Level View

Figure 20.1 illustrates a high-level view of our approach. First, we extract the computational resource requirements of different tasks in a network processor either by actual system measurements or through simulation. This collected information includes energy and execution latency values for each task on all processor types. This information is subsequently passed to the ILP solver, which determines the best processor matching the needs of the task to meet the performance deadlines while keeping the power consumption at the lowest rate. The ILP tool is provided with a pool of processors to choose from, where our goal in selecting the processor for each task is to minimize the energy consumption. Note that each processor can exhibit different characteristics in terms of performance, energy, temperature, area, and communication bandwidth supported.

TABLE 20.1 Performance Values of Security Engines with Different Processors

Task	Processor	Throughput
AES engine	100 MHz	1.28 Gb/s
	83 MHz	1.06 Gb/s
	58 MHz	0.39 Gb/s
RSA engine	250 MHz	80.7 Kb/s
	83 MHz	66.9 Kb/s
HMAC-SHA1	89 MHz	140 Mb/s
	83 MHz	130.5 Mb/s
RNG	200 MHz	6.40 Gbps
	83 MHz	2.66 Gb/s

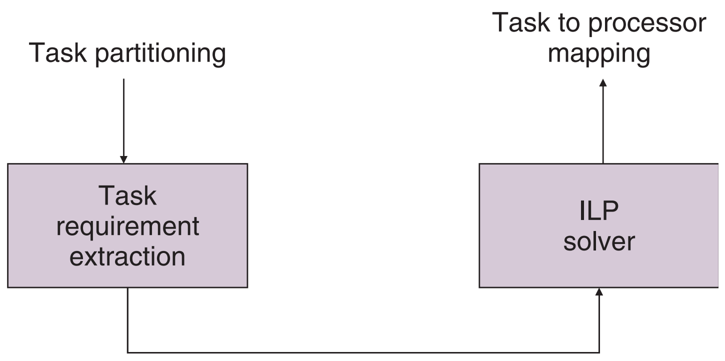

FIGURE 20.1 *High-level view of our approach.*

20.4 HETEROGENEOUS CMP DESIGN FOR NETWORK SECURITY PROCESSORS

20.4.1 Task Assignment

A widely used application model on a CMP-based network processor is to implement a pipeline where each packet is processed by multiple processors performing different tasks [26–28]. In this approach, packet processing tasks are assigned to a stage of the pipeline. Each of these stages is processed by a different processor of the CMP. We do not consider the task partitioning problem in this work; rather, we assume that a set of ordered tasks has already been implemented as a pipeline in the network processor.

As shown in Fig. 20.1, we first need to extract the task processing requirements for a given network system. We use profiling information to identify the computational requirements of each given task in the pipeline. While this part of the system has not been completely automated yet, we are able to generate close estimates and use them in our ILP formulation for preliminary tests.

In this representation, shown in Fig. 20.2, the execution of the network system is viewed as a series of *tasks* preformed on the packets. Once the task partitioning is known, we can execute each task on the available processor types. As a result, we

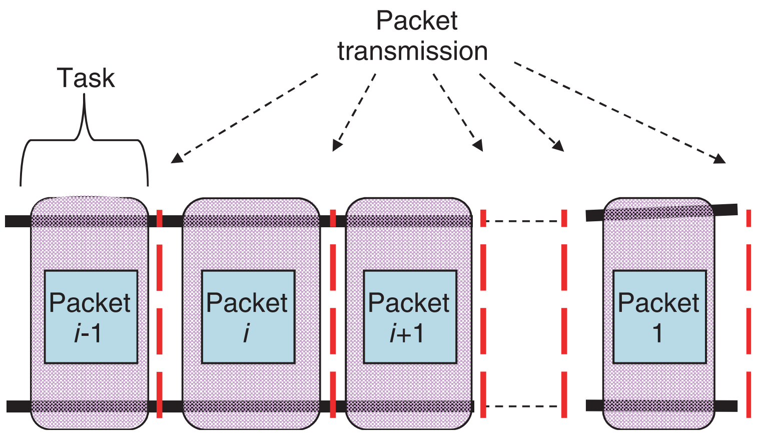

FIGURE 20.2 *Example network processor pipeline.*

obtain energy and execution latency values for each task on all processor types in our pool. Our ILP-based optimization approach operates on these values and selects the most suitable processor to run the task on. The goal of our ILP formulation, explained in the next section, is to select the processor type for each task to minimize the overall energy consumption.

20.4.2 ILP Formulation

Our goal in this section is to present an ILP formulation of the problem of minimizing the energy consumption of a given network security system by determining the optimal processor types that can be used in a heterogeneous CMP. Table 20.2 gives the constant terms used in our ILP formulation. We used the *Xpress-MP* [33], a commercial tool, to formulate and solve our ILP problem, though its choice is orthogonal to the focus of this chapter. In our baseline ILP formulation, we assume that our design is not limited with any area constraint, that is, we can allocate any type of processor to each task. Our goal with this ILP formulation is to select the most suitable processor for individual tasks.

Assume that we are given N number of tasks, t_i, where $1 \leq i \leq N$. Our approach uses 0–1 variables to select a processor for a given task among the processor pool, and at the end returns the overall task to processor mapping. In our design, we assume that we have a pool of processors $\text{Proc}_1, \ldots, \text{Proc}_P$, where frequency of Proc_i is higher than that of Proc_j if $i > j$. More specifically, Proc_P is the fastest processor with the highest area requirements, whereas Proc_1 has the lowest frequency and area. Execution latency of a task is expressed by $L_{t,p}$, which indicates the latency of executing task t on processor p. As explained earlier, latency information has been extracted using simulators and through actual implementations.

We use the $M_{t,p}$ variable to indicate that task t will be running on processor type p. More specifically,

- $M_{t,p}$ indicates whether task t is running on processor type p.

TABLE 20.2 Constant Terms Used in our ILP Formulation; These are Either Architecture-Specific or Program-Specific

Constant	Definition
N	Number of tasks
P	Number of processor types
$L_{t,p}$	Execution latency of task t on processor type p
$E_{t,p}$	Energy consumption of task t on processor type p
Area	Area allocated for processors
Area_p	Area needed for processor p
α	Weight of energy in the total cost
β	Weight of throughput in the total cost

Similarly, $T_{\max}$ is a variable that indicates the maximum execution latency of a given task. This can be expressed as

$$T_{\max} = \max_t L_{t,P}, \quad \forall t \text{ with } \mathrm{Proc}_P. \tag{20.1}$$

As explained earlier, we assume that network processor is implemented to process the packets in a pipelined manner. Network pipeline performance is usually measured by the throughput it achieves, which can also be expressed as the number of packets processed per second. Pipeline throughput is limited by the maximum latency among all the tasks. We can find the maximum latency among the tasks by comparing their latencies on Proc_P, as it is the fastest processor in the processor pool.

After describing the variables, we next give our constraints. Our first constraint is regarding the unique assignment of a task, that is, a task can be assigned to a single processor:

$$\sum_{i=1}^{P} M_{t,i} = 1, \quad \forall t. \tag{20.2}$$

In the above equation, i corresponds to the processor type and iterates over all the combinations. When executed on a slower processor, a task may have a longer execution latency compared to $T_{\max}$, which potentially can reduce the pipeline throughput. In order to prevent such cases, we need to make sure that maximum allocated time for any task is limited by $T_{\max}$.

$$\sum_{i=1}^{P} M_{t,i} \times L_{t,i} \le T_{\max}, \quad \forall t. \tag{20.3}$$

Meanwhile, we aim at reducing the energy consumption. To achieve this, we select an energy-efficient processor that does not degrade the pipeline throughput.

$$\mathrm{TE}_t = \sum_{i=1}^{P} M_{t,i} \times E_{t,i}, \quad \forall t. \tag{20.4}$$

In this expression, TE_t indicates the energy consumption of task t or the corresponding pipeline stage to process one packet. This energy will depend on the processor mapped to the task and the energy value of that processor.

Having specified the necessary constraints in our ILP formulation, we next give our objective function. We define our cost function as the sum of the task energies, which is given by $\mathrm{Energy}_{\mathrm{total}}$. Consequently, our objective function can be expressed as

$$\min \quad (\mathrm{Energy}_{\mathrm{total}} = \sum_{i=1}^{N} \mathrm{TE}_i). \tag{20.5}$$

To summarize, our processor selection problem can be formulated as "minimize $\mathrm{Energy}_{\mathrm{total}}$ under constraints (20.1) through (20.4)." It is important to note that this ILP formulation is very flexible, as it can accommodate different number of processors and tasks.

20.4.3 Discussion

Note that, in our ILP formulation, we employ performance and energy as constraints, whereas area, temperature, communication bandwidth, and other possible constraints are left out. For example, depending on the area constraint, it may not be possible to accommodate a certain processor mixture. Our ILP formulation, presented earlier, does not cover this constraint and similar ones. However, the ILP problem can easily be modified to include such constraints. Area constraint can be added by simply adding the areas of processors that are being used for tasks.

$$\text{Area} \geq \sum_{i=1}^{N} \sum_{j=1}^{P} M_{i,j} \times \text{Area}_j. \tag{20.6}$$

The right-hand side of the expression given above sums up the areas of the processors that are being used, and this sum should be less than the total area available. Note that we are not doing an exact placement within the available area, which would require further analysis.

To include area as one of the constraints, we also need to modify (20.1), which is used to find $T_{\max}$. This comes from the fact that we no longer know whether Proc_P will fit into the available silicon area with the rest of the processors. Our new $T_{\max}$ expression will consider all processors and all possible mappings. This can be achieved directly using (20.2) and removing (20.1) from the constraint set.

After removing (20.1), one can observe that the ILP tool will select the lowest frequency processors for all the tasks since our objective is to minimize the energy consumption. This will obviously increase the execution latency, which is very critical for network systems. To prevent such problems, we can use a weighted approach where we use both energy and throughput in our objective function. More specifically

$$\min \quad (\alpha \times \text{Energy}_{\text{total}} + \beta \times T_{\max}). \tag{20.7}$$

The first part of the above expression captured with a weight of α emphasizes energy, whereas the second part given with a weight of β stresses on the performance. Note that $T_{\max}$ gives the maximum latency of any one task in our pipeline. Assigning a very large value to β compared to α will try to first reduce the $T_{\max}$ value as much as possible. When $T_{\max}$ reaches the maximum value, then the ILP tool will exploit the energy reduction opportunities.

Note that so far we did not assume any limit on the number of processors in any processor type. Our ILP formulation can easily be modified to include processor count as a constraint as well; however, because of space limitation we do not go into details.

One can also optimize the placement of the processors within the CMP to minimize the associated communication overhead. Although not presented here, our formulation can easily be modified to reflect such a goal. For example, we can incorporate coordinates to processors and reduce the communication distances between the subsequent tasks. In our future studies, we plan to explore the aforementioned objectives.

20.5 EXPERIMENTAL EVALUATION

20.5.1 Setup

We tested our approach with 10 different network security system scenarios. We assume that, for each network system, there are multiple tasks required to be performed in a pipelined manner, as given in Section 20.4.1. As explained earlier, we assume that task assignment is already available, and we only select the processor type to run each of these tasks. The available processor types and their characteristics are listed in Table 20.3. The second column gives the IPC value of each processor type, while the third column shows the average energy consumption, and the last column shows the area required for the processor. Note that the values given in this table represent normalized values based on Alpha cores.

On the other hand, Table 20.4 lists the tasks along with their execution latency and energy consumption values. Note that the execution latency and energy values are obtained by running the given task on processors P_1–P_4. To compare the energy reduction brought by a heterogeneous CMP over a homogeneous CMP using multiple pipelines, we randomly selected 10 different subsets of the tasks in Table 20.5. The second column of the table lists the selected tasks. We performed experiments with four different optimization schemes for each pipeline in our experimental suite:

- *HM:* In this approach, we implement a homogeneous CMP using the same type of processors. We implement this approach within our ILP framework by adding

TABLE 20.3 Processors Used in Our Heterogeneous CMP and Their Characteristics

Processor	IPC	Energy	Area
P_1	1	3.73	1
P_2	1.3	6.88	2
P_3	1.87	10.68	8
P_4	2.14	46.44	40

TABLE 20.4 Tasks Used in This Study

Task	P_1		P_2		P_3		P_4	
	Latency	Energy	Latency	Energy	Latency	Energy	Latency	Energy
T_1	0.50	1.87	0.38	2.65	0.27	2.86	0.23	10.85
T_2	1.40	5.22	1.08	7.41	0.75	8.00	0.65	30.38
T_3	1.60	5.97	1.23	8.47	0.86	9.14	0.75	34.72
T_4	0.80	2.98	0.62	4.23	0.43	4.57	0.37	17.36
T_5	2.00	7.46	1.54	10.58	1.07	11.42	0.93	43.40
T_6	1.20	4.48	0.92	6.35	0.64	6.85	0.56	26.04
T_7	2.10	7.83	1.62	11.11	1.12	11.99	0.98	45.57
T_8	0.20	0.75	0.15	1.06	0.11	1.14	0.09	4.34

Execution latency and energy consumption values are obtained by running tasks on processors $P_1, \ldots, P_4$.

TABLE 20.5 Pipelines Tested

Pipeline	Tasks
Pipeline$_1$	T_1, T_3, T_4, T_6, T_8
Pipeline$_2$	T_2, T_3, T_4, T_5
Pipeline$_3$	T_1, T_4, T_6, T_7, T_8
Pipeline$_4$	T_2, T_5, T_7
Pipeline$_5$	T_3, T_6, T_7
Pipeline$_6$	T_2, T_4, T_6, T_7
Pipeline$_7$	T_1, T_2, T_4, T_7
Pipeline$_8$	T_1, T_3, T_5, T_6
Pipeline$_9$	$T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8$
Pipeline$_{10}$	T_1, T_3

an additional constraint that forces all processors to be the same type:

$$M_{t_1,p} = M_{t_2,p}, \quad \forall t_1, t_2, p. \tag{20.8}$$

This constraint makes sure that, if task t_1 is assigned to a certain processor type p, then the rest of the tasks should also be mapped to the same processor type.

- *HT:* This is the first ILP-based heterogeneous CMP strategy discussed in this chapter (Section 20.4.2).
- *HM+:* This is very much similar to HM except that it enforces the area constraint given in Section 20.4.3.
- *HT+:* This is an extension to the HT scheme wherein area constraints are also applied. This strategy is discussed in Section 20.4.3 in detail. Note that we set the default area available for the processors as 200 units. All the listed tasks in the pipelines need to be implemented within this area limit. Later, we also modify the default area to test the sensitivity of our approach. The ILP solution times for our approaches range from 0.5 to 15 s, with an average of 3 s, across all the test cases.

20.5.2 Results

We first evaluate and compare our approach HT to HM scheme for pipelines given above in Fig. 20.3. Each bar represents the normalized energy consumption of our approach (HT) with respect to the HM case. As can be seen from the graph (Fig. 20.3), on average, our approach reduces the energy consumption by 41% over the HM case.

We also performed experiments with area constraints enforced. As has been stated earlier, for the default case we assumed abundant area. The graph in Fig. 20.4 shows the experimental results for the HM+ and HT+ approaches. Each bar corresponds to the percentage reductions of HT+ over the HM+ case, respectively. We observe that the reductions vary between 7% and 48%. On average, our approach yields a 37% reduction in the energy consumption over the HM+ approach.

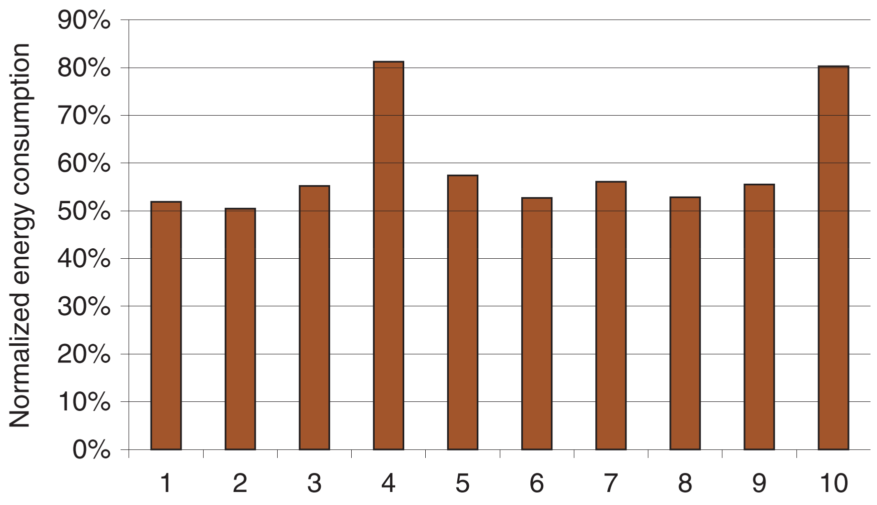

FIGURE 20.3 *Normalized energy consumption in our ILP-based HT approach over the HM approach.*

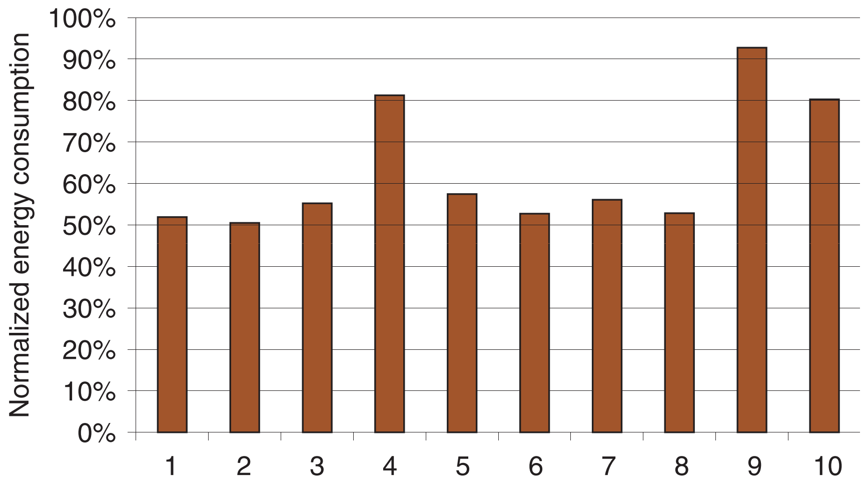

FIGURE 20.4 *Normalized energy consumption with the HT+ approach over the HM+ case.*

Recall that the pipelines given in Table 20.5 use a fixed area constraint of 200 in the default case. The graph in Fig. 20.5 shows the percentage reductions with different area constraints. In this figure, we specifically test the sensitivity of our approach to the area constraint.

As can be seen from this chart, energy savings are higher with a larger processor area. This follows from the fact that, with increased area, HM+ uses the fastest processor to increase the throughput. However, this also increases the wasted energy on the noncritical tasks using the same power-hungry processor. However, HT+ selects a power-efficient processor for the noncritical tasks while using a fast processor for the critical ones. One can also observe from these results that the reduction brought using HT+ approach over the HM+ case is 26%, on average.

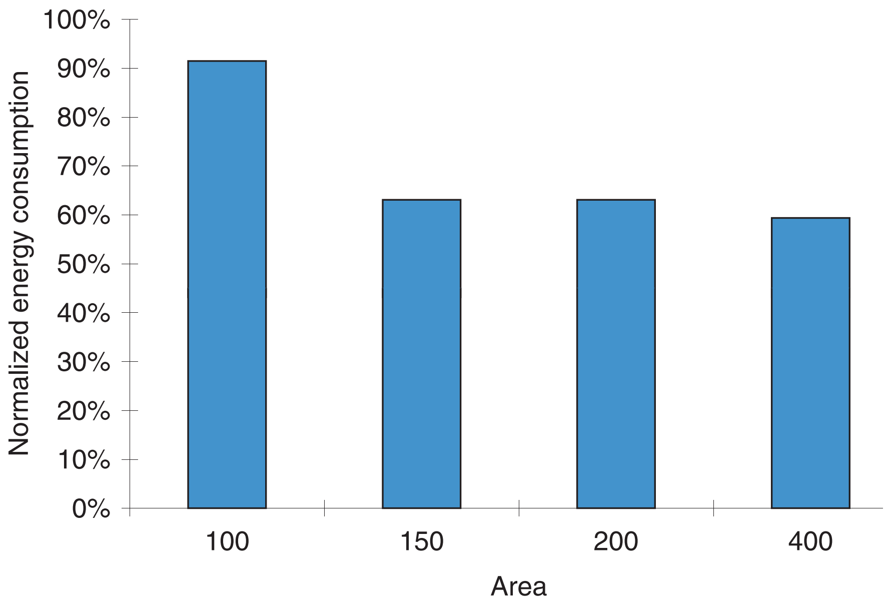

FIGURE 20.5 *Sensitivity of our approach to the total processor area available.*

20.6 CONCLUDING REMARKS

Growing importance of security within networking devices makes it imperative to consider techniques to optimize performance and power consumption of security tasks in network processors. Motivated by this observation, this chapter proposed and experimentally evaluated an ILP-based approach to use heterogeneous CMPs in network security processors. The goal was to use the most suitable processor for a given task as much as possible, thereby saving energy while not reducing the throughput. We tested our approach using synthetic task sets. Our experimental results indicate that heterogeneous CMPs reduce the energy consumption dramatically compared to homogeneous CMPs. We also found that the solution times taken by our approach were within tolerable limits for all the cases tested.

ACKNOWLEDGMENTS

This work was supported in part by Open European Network for High-Performance Computing on Complex Environments, the TUBITAK Grant 112E360, and a grant from Turk Telekom under Grant 3015-04.

REFERENCES

1. ITRS, “International technology roadmap for semiconductors.”
2. “Octeon by Cavium,” in http://www.cavium.com/OCTEON_MIPS64.html, 2014.
3. Endian, “Endian firewall macro x2,” in http://www.endian.com/en/products, 2008.

4. R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan, "Heterogeneous chip multiprocessors," *Computer*, vol. 38, no. 11, pp. 32–38, 2005.
5. R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA heterogeneous multi-core architectures for multithreaded workload performance," in *ISCA '04: Proceedings of the 31st Annual International Symposium on Computer Architecture*, p. 64, 2004.
6. H. Blume, H. T. Feldkaemper, and T. G. Noll, "Model-based exploration of the design space for heterogeneous systems on chip," *Journal of VLSI Signal Processing Systems*, München, Germany, vol. 40, no. 1, pp. 19–34, 2005.
7. S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai, "The impact of performance asymmetry in emerging multicore architectures," in *ISCA '05: Proceedings of The 32nd Annual International Symposium on Computer Architecture*, Madison, Wisconsin, USA, pp. 506–517, 2005.
8. F. Liu and V. Chaudhary, "Extending OpenMP for heterogeneous chip multiprocessors," in *Proceedings of International Conference on Parallel Processing*, Kaohsiung, Taiwan, p. 161, 2003.
9. L. Brisolara, S.-i. Han, X. Guerin, L. Carro, R. Reis, S.-I. Chae, and A. Jerraya, "Reducing fine-grain communication overhead in multithread code generation for heterogeneous MPSoC," in *SCOPES '07: Proceedings of the 10th International Workshop on Software & Compilers for Embedded Systems*, pp. 81–89. New York, NY: ACM, 2007.
10. S. L. Shee, A. Erdos, and S. Parameswaran, "Heterogeneous multiprocessor implementations for JPEG: a case study," in *CODES+ISSS '06: Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis*, pp. 217–222. New York, NY: ACM, 2006.
11. S. Gopalakrishnan and M. Caccamo, "Task partitioning with replication upon heterogeneous multiprocessor systems," in *RTAS '06: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 199–207. Washington, DC: IEEE Computer Society, 2006.
12. R. Grant and A. Afsahi, "Power-performance efficiency of asymmetric multiprocessors for multi-threaded scientific applications," *IEEE International Parallel and Distributed Processing Symposium*, p. 344, 2006.
13. M. Becchi and P. Crowley, "Dynamic thread assignment on heterogeneous multiprocessor architectures," in *CF '06: Proceedings of the 3rd Conference on Computing Frontiers*, Ischia, Italy, pp. 29–40, 2006.
14. R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction," in *MICRO 36: Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, San Diego, CA, p. 81, 2003.
15. R. Kumar, D. M. Tullsen, and N. P. Jouppi, "Core architecture optimization for heterogeneous chip multiprocessors," in *PACT '06: Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, Seattle, Washington, pp. 23–32, 2006.
16. J. Yan and W. Zhang, "Hybrid multi-core architecture for boosting single-threaded performance," *SIGARCH Computer Architecture News*, vol. 35, no. 1, pp. 141–148, 2007.
17. P. Lieverse, P. V. D. Wolf, K. Vissers, and E. Deprettere, "A methodology for architecture exploration of heterogeneous signal processing systems," *Journal of VLSI Signal Processing Systems*, vol. 29, no. 3, pp. 197–207, 2001.
18. S. Ghiasi, T. Keller, and F. Rawson, "Scheduling for heterogeneous processors in server systems," in *CF '05: Proceedings of the 2nd Conference on Computing Frontiers*, Ischia, Italy, pp. 199–210, 2005.

19. C.-P. Su, C.-H. Wang, K.-L. Cheng, C.-T. Huang, and C.-W. Wu, "Design and test of a scalable security processor," in *ASP-DAC '05: Proceedings of the 2005 Conference on Asia South Pacific Design Automation*, Shanghai, China, pp. 372–375, 2005.
20. L. Wu, C. Weaver, and T. Austin, "CryptoManiac: a fast flexible architecture for secure communication," in *ISCA '01: Proceedings of the 28th Annual International Symposium on Computer Architecture*, Göteborg, Sweden, pp. 110–119, 2001.
21. C.-H. Wang, C.-Y. Lo, M.-S. Lee, J.-C. Yeh, C.-T. Huang, C.-W. Wu, and S.-Y. Huang, "A network security processor design based on an integrated SOC design and test platform," in *DAC '06: Proceedings of the 43rd Annual Conference on Design Automation*, Anaheim, CA, USA, pp. 490–495, 2006.
22. DELL SonicWALL, "The advantages of a multi-core architecture in network security appliances," in http://www.sonicwall.com, 2008.
23. D. Tian and Y. Xiang, "A multi-core supported intrusion detection system," in *NPC '08: Proceedings of the 2008 IFIP International Conference on Network and Parallel Computing*, pp. 50–55, 2008.
24. A. Chonka, W. Zhou, K. Knapp, and Y. Xiang, "Protecting information systems from DDoS attack using multicore methodology," in *CITWORKSHOPS '08: Proceedings of the 2008 IEEE 8th International Conference on Computer and Information Technology Workshops*, Shanghai, China, pp. 270–275, 2008.
25. S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, and J. W. Lockwood, "Deep packet inspection using parallel bloom filters," *IEEE Micro*, Sydney, Australia, vol. 24, no. 1, pp. 52–61, 2004.
26. Y. Shoumeng, Z. Xingshe, W. Lingmin, and W. Haipeng, "GA-based automated task assignment on network processors," in *Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS'05)*, Fuduoka, Japan, pp. 112–118, 2005.
27. A. Mallik, Y. Zhang, and G. Memik, "Automated task distribution in multicore network processors using statistical analysis," in *ANCS '07: Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems*, Orlando, Florida, pp. 67–76, 2007.
28. S. Datar and M. A. Franklin, *Task Scheduling of Processor Pipelines with Application to Network Processors*. Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, Missouri, USA, 2003.
29. L. Yang, T. Gohad, P. Ghosh, D. Sinha, A. Sen, and A. Richa, "Resource mapping and scheduling for heterogeneous network processor systems," in *ANCS '05: Proceedings of the 2005 ACM Symposium on Architecture for Networking and Communications Systems*, Princeton, New Jersey, USA, pp. 19–28, 2005.
30. "Corezilla: build and tame the multicore beast," in *Proceedings of the 44th Annual Conference on Design Automation*, 2007.
31. B. Flachs, S. Asano, S. Dhong, P. Hotstee, G. Gervais, R. Kim, T. Le, P. Liu, J. Leenstra, J. Liberty, B. Michael, H. Oh, S. Mueller, O. Takahashi, A. Hatakeyama, Y. Watanabe, and N. Yano, "A streaming processing unit for a CELL processor," in *Digest of Technical Papers from Solid-State Circuits Conference*, San Diego, CA, USA, Vol. 1, San Francisco, CA, USA, 2005, pp. 134–135.
32. D. Pham, S. Asano, M. Bolliger, M. Day, H. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa, "The design and implementation of a first-generation CELL processor," in *Digest of Technical Papers from Solid-State Circuits Conference*, San Francisco, CA, USA, Vol. 1, 2005, pp. 184–592.
33. FICO, "Xpress-MP," http://www.dashoptimization.com/pdf/Mosel1.pdf, 2002.