Generalized Pattern Matching Micro-Engine

Yuanwei Fang, Raihan ur Rasool†, Dilip Vasudevan, Andrew A. Chien*

Dept. of Computer Science, University of Chicago, {fywkevin,dilipv}@cs.uchicago.edu

Dept. of Computer Science†, King Faisal University†, rrasool@kfu.edu.sa†

MCS Division*, Argonne National Laboratory*, achien@cs.uchicago.edu*

Abstract

Many big data applications including dictionary-based decoding, deep packet inspection, Bioinformatics (DNA Alignment), and JSON/XML processing depend on fast pattern matching – well-known as difficult to accelerate. We have designed a novel heterogeneous architecture, called the Generalized Pattern Matching micro-engine (GenPM), to accelerate FSM-based applications. GenPM balances programmability and performance, employing a novel micro-architecture and software interface. We implement and evaluate GenPM in a 32nm TSMC process using a Snort network monitoring workload. Results show 64-way, 16-step GenPM can reduce instruction count by >1800x, effectively eliminating instruction management as a bottleneck. Remarkably, this reduction produces >2400x performance increase and energy efficiency benefits of 980x. Less aggressive GenPM's, such as 8-way,16-step achieve 200-fold instruction reduction, 300-fold performance increase, and 170-fold energy efficiency benefit. GenPM approaches ASIC efficiency, while maintaining programmability.

1. Introduction

Endemic to the explosion of big data is the need to find patterns in a sea of unstructured, noisy, and complex data. Thus, pattern-matching applications are essential tools, and must operate at high speed and efficiency to be useful at "big data" scale. Canonically, pattern-matching uses regular expressions (RE) to express search patterns, and then extracts all sets of the data that match the regular expressions. For example, pattern-matching applied to network flows might look for attack patterns or applied to flow contents look for virus signatures (network monitoring and intrusion detection). Network flows at 100Gig comprise a Petabyte/day. In bioinformatics, a human genome is several gigabytes, but many plant genomes are much larger, and pattern-based search is a central computational tool (bioinformatics). Content search over XML/JSON databases also benefits from rich pattern-matching capabilities (data mining, high frequency trading, deep content search). High performance for such data intensive applications has been the subject of extensive research [1, 2, 3, 4, 5].

While here our focus is pattern-matching, our broader focus is the design of a new class of energy-efficient and programmable microarchitectures capable of servicing a wide variety of pattern matching applications. With the benefits of Dennard Scaling [6] fading, our approach is to increase both performance and energy efficiency with a customized architecture.

Many pattern-matching accelerators have been proposed, exploiting ASIC, FPGA, GPU or multi-core approaches [7, 8, 9, 10]. Recent publications also explore new algorithmic and software approaches [11,12], demonstrating the relevance and central importance of the problem. While each of these approaches improves performance, relative to a traditional single-core CPU, they are all far from the maximum achievable performance. We will show in GenPM how large this gap is. GenPM, a customized core that exploits local memory, SIMD operations, and complex instructions to accelerates pattern-matching on big data. Our results show that performance can approach that of ASIC's while maintaining programmability. Specific contributions include:

- Design of a novel micro-architecture for generalized pattern matching and ISA (GenPM),
- Implementation of GenPM in 32nm CMOS enabling rigorous timing and energy evaluation,
- Evaluation of GenPM performance using the RegEx software system with a standard SNORT workload, demonstrating 36x to as much as 2500x performance improvement(depends on GenPM configuration),
- Evaluation of GenPM energy use showing as much as 31x to 980x improvement (depends on GenPM config),
- Area and energy efficiency studies that show GenPM can scale to 2.6 Trillion DFAops/second in a full chip-scale design,

The remainder of the paper is organized as follows. Section 2 introduces key background, and Section 3 introduces the GenPM architecture and its use. Section 4 describes the evaluation and Section 5 shows detailed discussion with related work. Section 6 summarizes and suggests directions for future work.

2. Background

Deterministic Finite Automata(DFA) is a natural formalism for regular expressions and has a wide range of application.

Deep Content Inspection involves thorough searching of packets payloads against thousands of rules to identify intrusive or malicious behavior at wire speed. DCI systems employ REs or simple strings to express the patterns using DFAs due to their ability to do high speed matching in fast pace networks.

XML is currently the most popular format for exchanging and representing data on the web. As DFA takes constant amount of time to process one (SAX) event, several recent works employ it to represent queries [13].

For bioinformatics applications, DFA is used for Gene finding, Motif finding, protein secondary sequence prediction, splice site predictions, restriction site finding and generally in biological data mining [14].

3. Generalized Pattern Matching Engine

We describe microarchitecture and instruction set of the Generalized Pattern Matching Micro-Engine (GenPM).

3.1 RegEx on GenPM

Our prototype pattern-matching system uses the RegEx program as a front-end, uses its compiler to create efficient DFA tables which are then implementation by GenPM. First, RegEx Processor [15] converts the RE expression to DFA tables. Next, GenPM translator rewrites the DFA table generating a format specifically compatible with GenPM's matching unit. It takes the DFA from RegEx combining the acceptance and state.

3.2 Microarchitecture

GenPM executes normal instructions such as ALU, Load, Store, Branch, as well as special instructions for pattern-matching acceleration. It includes a multi-bank local memory that is used to store DFA tables (but can also be used for many other things) as well as a traditional memory hierarchy. The critical parameters for the GenPM architecture are **vector length**, **local memory parallelism**, and **DFA-steps**. Vector instructions are used to implement multiple DFAs process against one input steam.

GenPM retrieves string data from the main memory, and its Block Mover loads DFA tables from main memory into the local memory. The Block Mover is a DMA engine that efficiently transfers data between main memory and local memory, operating autonomously without stalling the main pipeline. It operates at low priority, stalling if memory bandwidth is unavailable. To begin processing a stream against a set of DFA's, a programmer initializes a vector register with the base addresses of a set of DFAs. At each step, the matching unit generates a set of next state addresses. The 6-stage GenPM pipeline is shown in Figure 1. Special registers and purpose are shown in Table 1.

3.2.1 DFA parallelism

GenPM exposes fine-grained DFA parallelism through a vector instruction interface proportional to vector length (or GenPM "width"). For each state transition, GenPM takes the DFA base address in the *GM_VEC_BASE* and calculates the target state addresses in parallel. This parallelism, combined with efficient state encoding and sequence (a pointer address rather than program counter), dramatically reduces instruction counts.

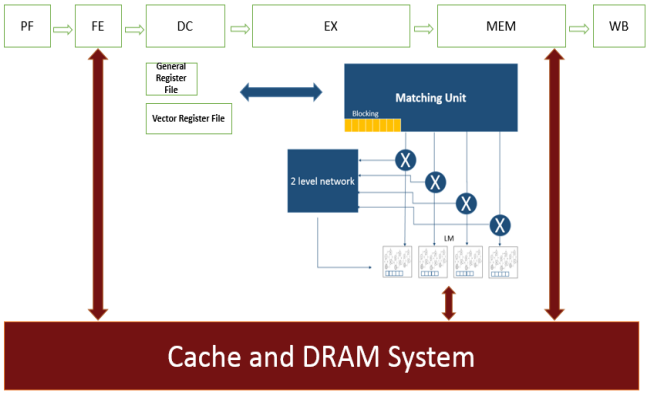

Figure 1. Microarchitecture of GenPM

Resource	Function spec.	Description
Local Memory	1MB--Local Memory	Local Memory for micro-engines. Store DFA tables
GM_VEC_STATE	1024bit--Vector Register	Store each DFA table's current state
GM_VEC_ACC	1024bit--Vector Register	Record acceptance for 8-64 DFA table for 1 -16steps
GM_VEC_BASE	1024bit--Vector Register	Base addresses for a set of 8-64 DFA tables
GM_VEC_BUF	1024bit--String buffer	Holds input string that need to be processed

Table 1. GenPM special registers

3.2.2 Matching unit, Multi-step

The matching unit implements parallel DFA state sequence and acceptance testing. It can advance a number of DFAs forward 1, 8, or 16 steps. In addition to implementing multi-step DFA sequence, it checks against acceptance states, flagging those DFA's that have accepted the input string. These matches are reflected as a vector of values, which is then parsed by software to give precise information (which DFA, exactly which point in the string) under software control.

3.2.3 Local memory

GenPM system performance also depends intimately on the local memory latency and parallelism. We studied single bank and multi-bank memories, for instance, an 8-wide GenPM can process 8 DFA tables simultaneously; given sufficient memory parallelism is available. If the local memory has 8 banks -- each with a read and a write port --

and there is no bank contention, minimum latency is achieved.

GenPM leaves DFA table allocation to software, and presumes non-conflicting layout can be achieved. If conflicts occur, performance is degraded, but hardware ensures correctness. Local memory addresses is embedded in the processors regular address space for convenient access. Each DFA is laid out contiguously in the address space with 256 entries (for every possible input) per state. We combine each state transition and accept rule with 16 bits. Thus, every DFA table is *N*512* bytes, N is the number of states in the DFA. We plan to study efficient compressed representations in future work.

3.2.4 Direct Network for Local Memory

Multi-bank memories can be built to match processor clock rates for small numbers of banks, but as the number of banks increases, network latency is a challenge. To avert any impact on typical case performance, we implemented a 2-level direct network of 8x8 switches for the 64-wide GenPM as shown in Figure *1*. Typical case for pattern matching achieves local, non-conflicting access.

3.3 GenPM Instruction Set Architecture

GenPM extends a simple 16/32-bit RISC instruction set, adding ten special 32-bit instructions to accelerate pattern-matching applications. We outline the key instructions**.**

GMVSNEXT which provides parallel processing among DFA tables and multi-step pattern matching. It receives 3 input parameters: buffer pointer, buffer index, and number of steps to process. Buffer pointer points to the latest position in string buffer vector register, buffer index chooses which buffer vector to read from after employing "double buffering" scheme.

GMVSACC tests whether there are acceptances among DFA tables, if accept then it returns the number of acceptances, and the program goes into the hit handler.

GMVSCONT takes out the value which records the matching length for current matching string.

GMLD,GMST loads/stores data block from/to the main memory to/from the local memory.

GMBUFLD,GMBSLD loads input string and the base address of each DFA table into corresponding vector register.

GMSTRIDEBACK retrieves DFA states before current multi-step transitions when GenPM hits an accept rule in multiple step process. This enables recomputing the fine-grained state transitions during the last multi-step procss and tell the exact matcing location.

GMCHECK check the GenPM's status register of multi-cycle executions for futher process.

GMVSCLEAR reset all the GenPM registers to zero.

4. Evaluation

We evaluate the performance and energy efficiency of GenPM, comparing to a 32-bit, 6-stage in-order single issue RISC processor (RISC32). We implemented GenPM micro-engine based on RISC32 using Synopsys CAD flow using a 32-nm TSMC library. Local memory energy models are based on CACTI 6 [16]. We integrate MARSSx86[17] and DRAMsim2[18] into our memory system model.

RegEx software is used with test patterns from SNORT [19] and real network traces. We divide Snort set into those requiring regular expressions and use RegEx to generate DFA tables. DFA's small enough to fits into a local memory bank are chosen. Detail about test rules see Appendix A.

4.1 Hardware Configuration

GenPM and baseline RISC processor has a 32KB 8-way L1 instruction cache and a 24KB 6-way L1 data cache, an 8-way 512 KB L2 cache and a 4GB DDR3 DRAM system. The total local memory capacity for GenPM is 1MB. GenPM designs vary the vector length (GenPM width), the number of local memory banks and multi-step length.

4.2 Metrics

We define energy efficiency as the total energy consumed for a given workload and operations/joule.

$$Workload/energy = Throughput/Watt$$
$$=Num\_DFA \times processed\_characters/Joule$$

We define system throughput as DFA steps per second that is, the full chip capacity computing ability to process the number of input characters per second for all parallel DFA matching.

$$Throughput = Num\_DFA \times Processed\_characters/execution\_time$$

4.3 Performance

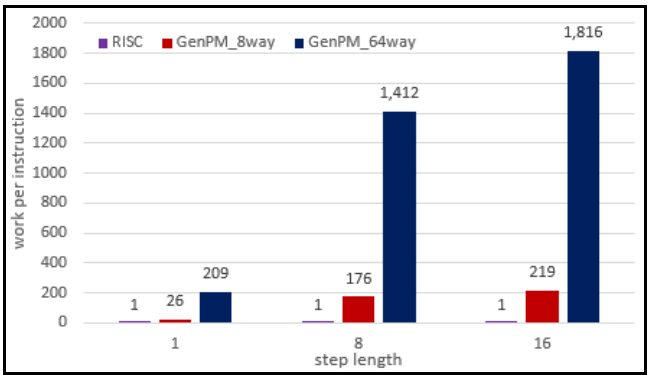

Figure 2. GenPM work per instruction for 10KB trace

We measured the kernel instruction count of Simple DFA_RE in GenPM and RISC32 with a 10KB input network trace and 64 Snort patterns. Figure 2 shows the work per instruction for GenPM with (8 or 64) local memory banks and same vector length with (1, 8, or 16-step) operations.

The result suggests that GenPM can significantly reduce instruction count. Even GenPM's instruction count for 8-way GenPM with 1-step execution is 26x times lower than RISC baseline. If no local memory contention arises, GenPM achieves 8 instructions overhead per vector/multi-step operation. Because per Amdahl's law this overhead limits performance, increasing steps/instruction can further increase single stream performance towards the hard limit of 1 character per cycle. In the most aggressive 64-way, 16-step GenPM configuration (with 64-bank local memory), achieves 1800X fewer instructions than the baseline RISC32.

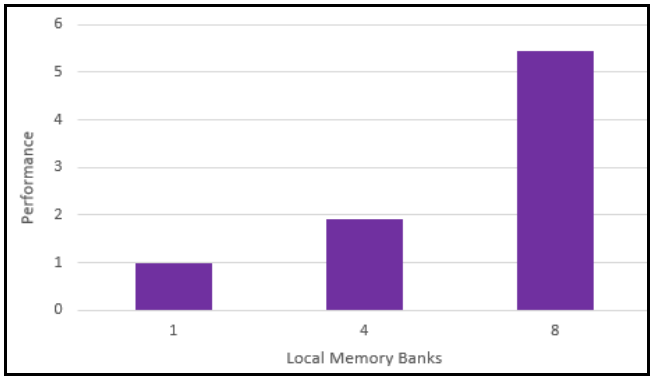

***Figure 3.Performance* impact of memory parallelism on GenPM with 8-step and 8 vector length**

Exploiting enough parallelism (multi-local memory banks), further improves the performance (see Figure 3), with benefits as large as an additional 5x. Figure 3 shows the relative performance of GenPM with 1, 4, or 8 local memory banks on an 8-way (vector length), 8-step GenPM. By employing multi-step operation, GenPM reduces while-loop condition check cost as well as *gmvsnext* instruction count. Figure 2 also shows the instruction benefit of multi-step operation length over different levels of GenPM DFA parallelism.

The total execution time includes the scalar instructions and the multi-step (multi-cycle). Figure 4 shows the speedup of GenPM with different configurations and extraordinary 2500x speedup achieved.

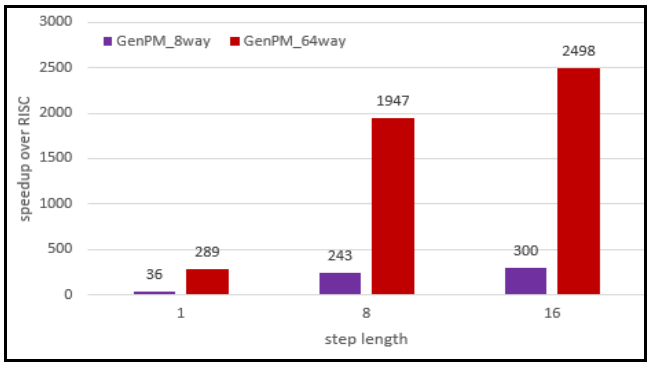

Figure 4. GenPM speedup (10KB trace)

4.4 Energy and Power Analysis

We compare GenPM and RISC32 with the same workload (10KB network traces, 64 Snort patterns), estimating energy and power of GenPM at 1GHz operation. The results (see Figure 5) show GenPM energy efficiency improvements from 31x to over 980x. The reduction comes both from a reduced runtime (less leakage energy), and via reduction of instructions and instruction fetch energy.

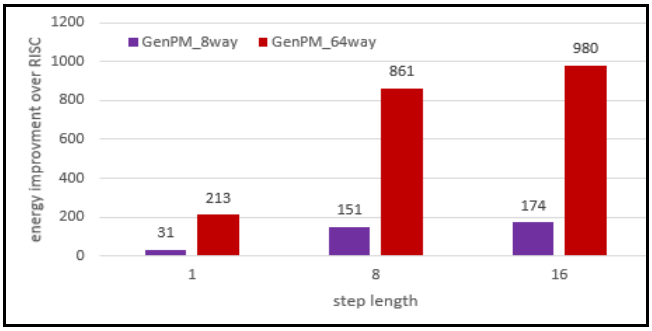

Figure 5. GenPM energy efficiency over RISC32

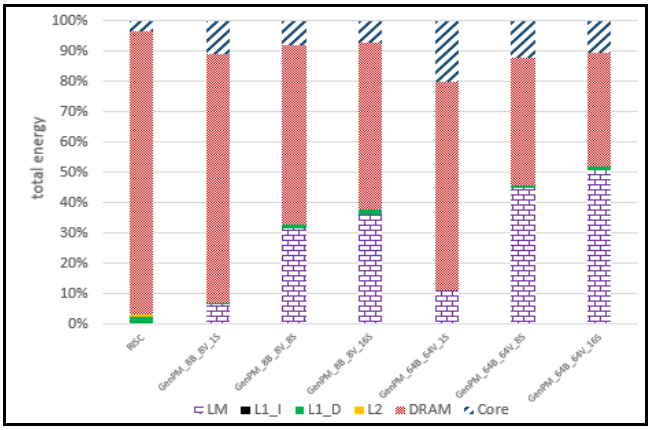

Figure 6. GenPM and RISC energy

Figure 6 shows the proportion of energy for GenPM (Banks,Steps,Vectors) configurations. GenPM's efficiency focuses the energy spent on the most valuable DFA work, the data memory references, reducing the other elements to increase energy efficiency. Figure 7 shows the power detail of GenPM, plotted in milliwatts. Even the most aggressive 64-way, 16-step GenPM consumes only 1 watt, yet delivers the performance of 16 Intel Ivy Bridge cores at 2.7GHz.

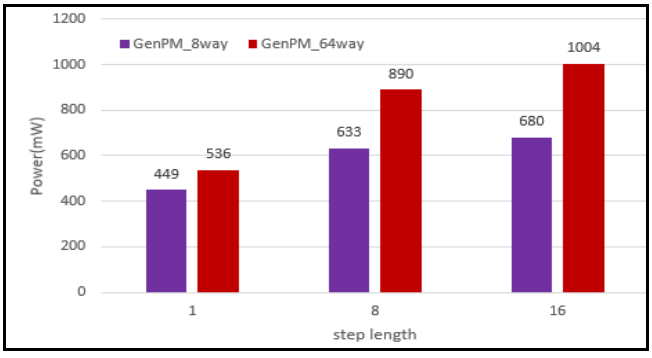

***Figure 7.* GenPM Power (various configs)**

4.5 Throughput and Area

GenPM-core die areas without memory are shown in Table 2. Most of the die area of GenPM core goes to network interconnection and vector registers. Figure *8* shows GenPM throughput and Figure 9 shows throughput-rate/Watt for different GenPM configurations. A single GenPM core can achieve >35GigaOps/sec; scaled up to a 75W chip, this is 2.6 trillion DFA ops/second to meet the needs of exponentially growing big data.

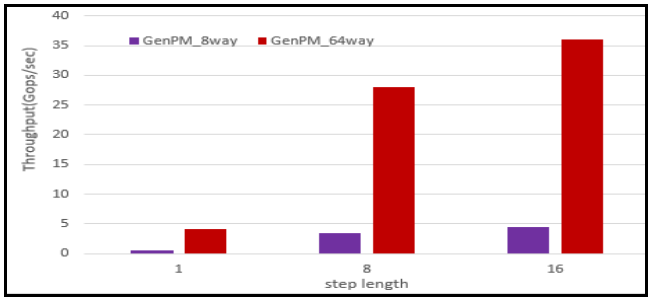

Figure 8. GenPM Throughput throughput-rate/Watt

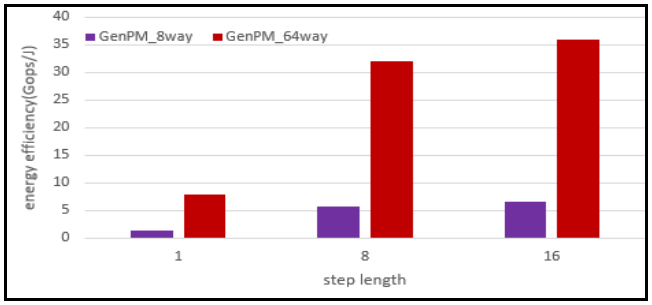

Figure 9.Throughput-rate/Watt

System	Process(nm)	Core area(mm^2)
RISC32	32	0.034
8-wide GenPM	32	0.223
64-wide GenPM	32	0.571

Table 2. Core silicon area

5. Discussion and Related Work

The GenPM design is a part of the 10x10 project [20 21], an ambitious effort to design general-purpose processors with much higher energy efficiency from collections of highly customized cores. GenPM is one of the micro-engines that would be combined as one of the 10, above a shared memory system.

To understand GenPM in context, we compare to performance and energy efficiency results to several prior, scaling for process differenes. The results are summarized in Figure 10.

ASIC Brodie et al [9] 's ASIC design in 65-nm process was projected to achieve a string line rate at 16Gbps, 500MHz clock frequency in 200mm^2 die area and 12KB RAM/engine. We estimate their design achieves throughput per Watt of 42Gops/J (detail see Appendix B). We scale to 32 nm process, so 168Gops/J (throughput per Watt)

GPU Vasiliadis et al [7] implemented a multiple input parallel pattern matching algorithm on Geforce GTX480. Full GPU card capacity throughput is 6Gops/s, and the power is 250W. Therefore, the throughput per Watt is 0.024Gops/J. Scaled to 32 nm, the GPU achieves 0.048Gops/J (throughput per Watt).

CPU Intel's HyperScan solution [22] for DPI application on 2.7GHz Intel Xeon E5-2600 (16 threads, 8 cores) achieves the throughput 134Gbps/8 = 16.75Gops/s in 130W. It achieves 0.13Gops/J (throughput per Watt)

Network Processor The IBM Power Edge of Network processor [23] in 45-nm process with 8 regular expression accelerators can achieve throughput 72Gbps/8 = 9Gops/s in 20W. Scaled to 32nm, IBM PowerEN achieves 0.9Gops/J (throughput per Watt).

GenPM The throughput of a 64 wide-16 step GenPM is 36Gops/s and with power of 1004mW. The throughput per Watt is 36Gops/J. Our design generate from high level architecture specification, so an optimized design could easily double GenPM's score at 72Gops/J (throughput per Watt).

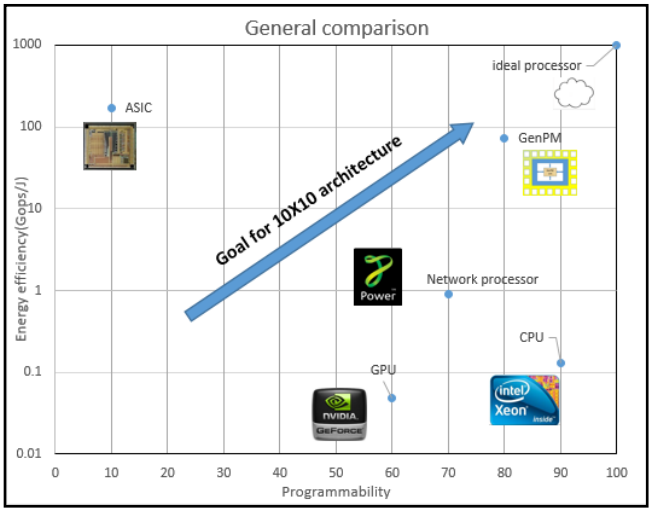

Figure 10. Energy Efficiency and Performance of Various Approaches

Figure 10 illustrates the performance-programmability space for FSM-based applications. While, GenPM has high energy efficiency while preserving a high degree of programmability.

6. Summary and Future Work

GenPM is a novel micro-architecture for a broad domain of pattern-matching applications. Our design and detailed evaluation show dramatic performance and energy efficiency improvements compared with FPGA, GPU, and CPU approaches, and are even comparable to ASIC systems. This enables extraordinary efficiency for

generalized pattern-matching applications in a flexible programmable system.

Promising avenues for future study include study with more advanced processes (a 7-nm process model), evaluation with additional workloads, further optimization based on DFA compression, and study of larger systems - multiple GenPM-cores.

7. Acknowledgement

This work is being supported in part by the Defense Advanced Research Projects Agency under award HR0011-13-2-0014, a gift from Agilent, and the generous Synopsys Academic program. We thank Tung Hoang and other members in LSSG for their valuable suggestions.

References

[1] Lin P C, Lin Y D, Lee T, et al. *Using string matching for deep packet inspection*. IEEE Computer, 2008, 41(4): 23-28.

[2] Bruno N, Koudas N, Srivastava D. *Holistic twig joins: optimal XML pattern matching*, Proceedings of the 2002 ACM SIGMOD international conference on Management of data. ACM, 2002: 310-321.

[3] Chen L, Lu S, Ram J. *Compressed pattern matching in DNA sequences*, Computational Systems Bioinformatics Conference, IEEE, 2004: 62-68.

[4] Dargham J, Al-Nasrawi S. *FSM behavioral modeling approach for hypermedia web applications: FBM-HWA approach*, AICT-ICIW'06, IEEE Press.

[5] Barford L A. *Parallelizing small finite state machines, with application to pulsed signal analysis[C]*//Instrumentation and Measurement Technology Conference (I2MTC), 2012 IEEE International. IEEE, 2012: 1957-1962.

[6] Frank, Dennard, Nowak, et al. *Device scaling limits of Si MOSFETs and their application dependencies*. Proceedings of the IEEE, 2001, 89(3): 259-288

[7] Vasiliadis G, Polychronakis M, Ioannidis S. *MIDeA: a multi-parallel intrusion detection architecture*, Proceedings of the 18th ACM conference on Computer and communications security. ACM, 2011: 297-308.

[8] Yang and Prasanna. *High-performance and compact architecture for regular expression matching on FPGA*. IEEE Trans. on Computers, 2012, 61(7): 1013-1025.

[9] Brodie B C, Taylor D E, Cytron R K. *A scalable architecture for high-throughput regular-expression pattern* matching, ISCA 2006, CAN 34(2): 191-202.

[10] Scarpazza D P, Russell G F. *High-performance regular expression scanning on the Cell/BE processor*, 23rd Intl Conf on Supercomputing. ACM, 2009: 14-25

[11] Mytkowicz T, Musuvathi M, Schulte W. Data-Parallel Finite-State Machines[J]. ASPLOS 2014.

[12] Zhao Z, Wu B, Shen X. Challenging the embarrassingly sequential: parallelizing finite state machine-based computations through principled speculation[C]//Proceedings of the 19th international conference on Architectural support for programming languages and operating systems. ACM, 2014: 543-558.

[13] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman, *"Efficient Pattern Matching over Event Streams"*,SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada

[14] Mulder, M. ; Grand Valley State Univ., Allendale, MI ; Nezlek, G.S,*Creating protein sequence patterns using efficient regular expressions in bioinformatics research*, 28th International Conference on Information Technology Interfaces,207 - 212 2006.

[15] http://regex.wustl.edu/index.php/Main_Page

[16] Thoziyoor S, Muralimanohar N, Ahn J H, et al. CACTI 5.1, HP Laboratories, April 2008.

[17] Patel A, Afram F, Ghose K. Marss-x86: A qemu-based micro-architectural and systems simulator for x86 multicore processors[C]//1st International Qemu Users' Forum. 2011: 29-30.

[18] Rosenfeld P, Cooper-Balis E, Jacob B. Dramsim2: A cycle accurate memory system simulator[J]. Computer Architecture Letters, 2011, 10(1): 16-19.

[19] SNORT: The Open Source Network Intrusion Detection System. http://www.snort.org

[20] Borkar S, Chien A A. *The future of microprocessors*. Communications of the ACM, 2011, 54(5): 67-77.

[21] Chien A A, Snavely A, Gahagan M. *10x10: A general-purpose architectural approach to heterogeneity and energy efficiency*. Procedia Computer Science, 2011, 4: 1987-1996.

[22] http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/160gbps-dpi-performance-using-intel-architecture-paper.pdf

[23] Golander A, Greco N, Xenidis J, et al. *IBM's PowerEN developer cloud: Fertile ground for academic research[C]*//Electrical and Electronics Engineers in Israel (IEEEI), 2010 IEEE 26th Convention of. IEEE, 2010: 000803-000807.

Appendix A

Detail about Snort rules used in the performance and energy evaluation. Note that the transition to default state is not counted as a transition.

Rule Name	States	Transitions	Rule Name	States	Transitions
1	17	51	33	29	57
2	24	6144	34	23	5888
3	27	6912	35	23	5888
4	20	5120	36	28	7168
5	8	2048	37	27	6912
6	13	3328	38	27	6912
7	20	5120	39	23	5888
8	29	57	40	24	3835
9	14	45	41	20	5120
10	21	42	42	21	5376
11	26	625	43	22	54
12	26	154	44	16	4096
13	22	5632	45	29	7424
14	24	6144	46	20	5120
15	26	6656	47	20	5120
16	32	107	48	23	45
17	20	5120	49	23	5888
18	32	8704	50	17	4352
19	8	2048	51	30	7680
20	13	3328	52	14	45
21	20	5120	53	21	42
22	21	5376	54	16	39
23	16	4096	55	10	67
24	23	5888	56	16	39
25	17	4352	57	16	39
26	14	3584	58	12	31
27	30	7680	59	19	46
28	20	5120	60	24	1575
29	32	178	61	18	785
30	23	1928	62	22	5632
31	17	51	63	24	6144
32	20	5120	64	27	6912

Appendix B

We estimate the ASIC throughput per Watt as follows: Each engine, which is 0.19mm^2, process 2 pattern with one memory access on average according to the published paper. Thus, the chip capacity throughput is $2 \times \frac{200 \times 90\%}{0.19} \times \frac{16G/s}{8} = 3790G/s$. If per RAM access energy is 0.12nJ, then under 500MHz the dynamic memory power is 30W. Estimating logic dynamic power doubles memory power produces throughput per Watt of 42Gops/J. We scale to 32 nm process, so 168Gops/J (throughput per Watt)