ISBN 978-981-11-3671-9
Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)
Beijing, 25-27 June, 2017, pp. 1-7
doi: 10.18178/wcse.2017.06.001

The Advantages and Challenges of Spatial Computation

Jian Weng, Yanan Lu, Leibo Liu[+], Shouyi Yin, and Shaojun Wei

Institute of Microelectronics, Tsinghua University, Beijing 100084, China

Abstract. Recent years witness great interest to extract more performance and energy efficiency from spatial computation which gains great reputation on various domains, ranging from signal processing to high-performance computation. Its success does not only rely on the performance boosting, but also energy efficiency which is a severe problem for the popular portable and wearable devices. Spatial computation has gradually become a part of the mainstream computing infrastructure. This paper offers an observation on the issues of processors, reviews some recent researches, discusses the advantages of spatial computation on flexibility, utilization, scalability and energy efficiency, as well as the challenges from programmability, compatibility, locality, etc.

Keywords: processor, spatial computation, performance, energy efficiency, etc.

1. Introduction

As the requirement of applications growing aggressively, computation units' characteristics became hot topics. Since the born of processors, performance always gets its popularity, even today, performance is the most significant quantification of a processors, both in academia and commerce. Under the guidance of Moore's Law, the performance is doubled every 18 months. From 1980s to 2004, the performance of a single core processor increased approximately 1.5 times per year [1]. At the same time, programmers and designers did a lot of effort to modify the software and hardware architectures to accelerate the applications. The revenue firstly came from the high-performance computation domain, in which most tasks showing explicit parallel potential, but not covering all the general purpose applications. Single core processor naturally fits for the sequential computation model, even employed some parallel technologies, e.g. pipeline, superscalar, but there are still some insurmountable troubles, e.g. design complexity increased.

A more powerful processor brings the applications a lot of benefits, but the size shrinking (e.g. more intensive resources) and higher performance (e.g. higher frequency) also mean more dedicated design and much larger power per unit area. For single core, the best way to improve performance is to increase the frequency, but empirical power formula indicates that the power consumption is proportional to the working frequency. Smaller size and higher power contribute to the thermal problem, which challenges the design of cooling system for massive computation device or station, and energy budget for portable devices is another challenge. Until 2012, the smart phone's sales are over 650 million and tablet's are over 150 million while traditional PC's are approximately 380 million [2], more and more situations which are constrained by both energy budget and power dissipation ask for more performance but lower power computation components.

Nowadays, the working frequency of the Intel mainstream products are approximately 3.5 *GHz*. And as a consequence, the memory accessing frequency of the operation increases too. With the technology progress evaluating, the secondary memory is not growing as fast as the processor and primary memory [3], leading to a larger and larger gap between them. For some applications, the memory accessing latency deteriorates the performance a lot.

[+] Corresponding Author;
Email: liulb@tsinghua.edu.cn.

Multicore is a mainstream solution for both the performance request and power dissipation issue. Performance does not only come from the increase of working frequency any more. Several cores integrated inside one chip by some other periphery connection architecture alleviates the power dissipation hot spot and the distribution of computation resources to several relative lightweight cores reduces the complexity of the circuits design [4]. Lower frequency also alleviates the memory accessing problem. And a multicore processor naturally explores the instruction-level parallelism (ILP) and thread-level parallelism (TLP). If the application involving a parallel memory accessing pattern, data-level parallelism (DLP) can be extracted. For example, GPU employs the single instruction multiple data (SIMD) strategy to accelerate the image processing applications and some scientific computation tasks which contains large massive regular data operations. But this does not always come up to our expectations. Some troubles with single core processor is just mitigated by the multicore programming model instead of eliminating.

According to the Amdahl's Law [5], even the number of cores closes to infinite, the performance will always be constrained by the sequential part of the application. As equation (1) showing, where p is the fraction of code can be accelerated, it can hardly be close to 1 even though the speed-up s is been pushed to infinity. The integral speed-up is limited by the sequential part. In a real-world application, the parallelism is always finite and limited. Finally, the sequential part is still a bottleneck. Furthermore, as the performance increased, these centralized computation models will face severe memory accessing problems. Fortunately, another stream of the computation architecture, called *spatial computation*, provides some solutions. In the next sections, we'll take an observation of the researches about the spatial computation.

$$S_{latency}(s) = \frac{1}{(1-p) + \frac{p}{s}} \tag{1}$$

2. Spatial Computation

Spatial computation is a paradigm that breaking the application's dataflow into regions, and these regions are mapped to some subset of the hardware resources, including functional units, interconnection network and storage, in the form of producer-consumer pipeline [7]. Spatial computation has some similarity with the multicore models. Both of them have more than one core, which is essentially paralleled. But there is a huge difference between them in computation resource organization. Multicore processor can be seen as a cluster of single cores, symmetry or asymmetry. And every core inside is a heavy or relative heavy core comparing to the spatial computation. The most special characteristic is that the spatial computing architecture usually implements the communication of different cores by point to point communication while some of the implementations also supports the shared memory like conventional processors. As shown in Fig. 1, it illustrates a kind of spatial architecture, coarse-grained reconfigurable array (CGRA). It shows the connection network and the details of the micro architecture of its processor element (PE).

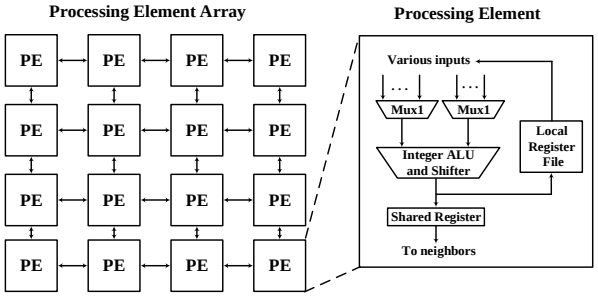

Fig. 1: A kind of spatial computing architecture, coarse-grained reconfigurable array (CGRA).

The example of CGRA indicates two characteristics of the spatial computing architecture: 1) no centralized control [6] (there're also some centralized controlled) and 2) lightweight execution units. Noticed that in the figure, it does not illustrate the memory architecture and that will be discussed in a follow-up section. But taking a glance of the register file in the PE architecture, there is a local register file instead of a set of centralized one. Next we'll discuss some advantages of the spatial computing architecture comparing with conventional ones. In Fig. 2(a)(c), it shows the patterns of a program, totally parallelizable and loop-data-

dependent, in which the circles with X_y represents instruction X of iteration Y, and the squares with x_y represents data read by instruction X_Y [7]. Fig. 2 (b) illustrates the computation organization of multi-instruction multi-data (MIMD) corresponding to the totally parallelizable situation. Fig. 2(d) illustrates the spatial computation corresponding to the situation of loop-carried dependence on instruction *A*, and the dependence is implemented by a register.

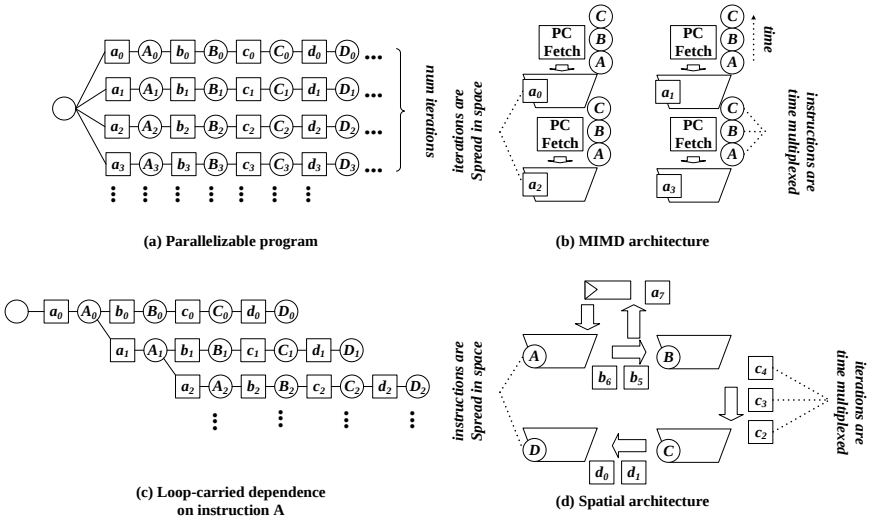

Fig. 2: An over view of the spatial computation [7].

2.1. Implementations of spatial computing architecture

2.1.1. DySER

Dynamically Specialized Execution Resource (DySER) [9] is a hybrid coarse-grained reconfigurable array aiming to improve both the performance and energy efficiency for general purpose processors in both sequential and parallel applications. DySER applied a dynamically specialized datapath to a conventional pipelined processor by a circuit-switched network. The network can be reconfigured, and it contains the hardware connection from the inputs to the functional units, point-to-point connections between the units, and to the outputs. DySER also contains a compiler to mapping the applications to functional units. The major function of the compiler is slicing the applications which require to be accelerated. Incorporation of the conventional processor and spatial architecture can effectively balance the sequential and parallel applications, offering a more flexible programming model, and provides more energy efficiency than pure conventional processors. But it also suffers the advantages of the energy efficiency of the spatial part for that the conventional processor cannot be shut downed while the workloads are all in the DySER blocks [4]. In [9], they evaluated a 64-functional-unit DySER block with dual-issue out-of-order processor on PARSEC, SPEC and Parboil benchmarks, obtained a 2.1x speedup and 40% energy reduction.

2.1.2. Wavescalar

The Wavescalar is a general purpose dataflow processors without the program counter (PC), aiming to develop a decentralized and scalable processor to obtain the performance of superscalar processors [10]. Without PC, the instructions in Wavescalar architecture are dynamically scheduled by dataflow instead of control flow. But unlike other static-dataflow architecture, Wavescalar's dynamic dataflow is similar to the out-of-order superscalar processor, but currently, it not supports the control speculation [4].

An implementation of Wavescalar architecture contains an array of PE tiles and connected to neighbours as an execution units cluster without long wires. It's a decentralized, uniform structure centring 4 identical domains which including 8 PEs. Every cluster contains a 4-banked L1 cache, and wave-order memory interface hardware. And between the clusters, there is switch network for communication. Inside the PE is five stage pipeline, containing one ALU and input and output networks which will be used to communicate with the neighbouring PEs after an instruction execution (there's no local register state).

2.1.3. TRIPS and TFlex

TRIPS is the first implementation of explicit data-graph execution (EDGE) processor [11][12]. It contains a 4x4 mesh grids of execution tiles (single-issue pipeline, a bank of 128 reservation stations, an integer

unit, a floating point unit, and an operand router) with 4 bank-register files tiles each including 32 registers, 4 D-cache and 4 I-cache banks (for increasing the memory bandwidth) serve the PE array, and there's a global control unit carrying the protocols of fill, flush, commit operations, which containing a separate I-cache. All these components are connected by the micro-network handling the transmission of operands between instructions, fetching instructions from the instruction tiles, and some control flow message from the global control units. According to [13], TRIPS classified as control flow/dataflow class models. The compiler of TRIPS plays a significant role which partitions the application codes to instruction hyperblocks and decides the placements of them to mitigate the communication delays. Dependent instructions will be placed near in space. And the block-atomic execution also helps TRIPS to be compatible to high level programming languages, e.g. C/C++. The inter-blocks are control flow based and the intra-block instructions are dataflow mapped, so it's a combination of the control flow and data flow model. In the block level view, the block is a "megainstruction" which is required to be fetched and executed as one.

TFlex [14] use the same ISA as TRIPS, obtaining a 42% boost in performance and 2x power-efficiency on average. Not like TRIPS the TFlex allowing the mapping algorithm to do a more accurate mapping according to the dataflow, and occupying adaptive hardware configuration [15].

2.1.4. TIA

Triggered Instruction Architecture (TIA) [7] is a distributed controlled PE array, the on-chip connection is latency-intensive channel which is the foundation of the flexible architecture, which is similar with the architecture shown in Fig. 1. But the microarchitecture of the TIA PE is not similar as it shown. The PE's datapath is lightweight and limited-complex, and the architectural state contains the following elements [7]: 1) data registers (R/W), usually, the configuration is 8; 2) predicate registers (R/W) which values are used as the trigger of the execution of instructions; 3) input-channel head elements (R-only); 4) output-channel tail elements (W-only).The data transmitted among PEs are tagged. The data tag combined with the predicate registers values constitute the trigger for the fetching and execution of instructions. The trigger eliminates the program counter in conventional architectures and naturally enables the out-of-order execution.

TIA PEs are autonomous scheduled by a scheduler inside each PE. The scheduler computes the predicate register values, channel tags, and channel validity values with the instruction trigger fields to schedule the instructions inside the instruction pool. Matched instruction will be fetched to the datapath for execution. There are usually 16 instruction entries in each PE. And in the circuits level view, the scheduler is only a tree of AND gates, so the scheduler computation will never become the critical path or consume too much energy.

Comparing to the conventional general-purpose processor, TIA gained 8x area-normalized performance. And comparing to the PC-based processor, 2x performance is obtained, in which 62% and 64% of the static and dynamic instructions are removed from the critical paths, respectively.

2.1.5. REMUS

REMUS [16] is a processor targeting on multiple-standard video decoding, and aiming to performance and energy-efficiency. Fig. 3 is the architecture of REMUS PEA and PE. The PEA in REMUS is called reconfigurable processing unit (RPU). Each RPU contains 4 slices of reconfigurable cell array (RCA) and each RCA contains 8x8 reconfigurable PEs as shown in the left of Fig. 3. RCA is independent to each other which offering a RCA level parallelism and certain RCAs can be shut down for a lower power consumption. The intra-RCA connection is called line-switched mesh connect (LSMC), which the PEs are organized in a line-to-line manner, i.e., each PE could be connected to any PEs in the adjacent upper and lower lines and the last line connects to the first line for iterative operations [16]. The inter-RCA is connected by a pair of I/O buses. There is only one line of PEs connected to the I/O bus which is determined by the configuration information. Comparing to a conventional full mesh, this connection topology decreases the complexity of interconnection implementation.

Right part of Fig. 3 is the micro-architecture of a PE, which contains an Arithmetic Logic Unit (ALU), input and output registers (A_REG, B_REG, C_REG), and temporary result register (T_REG) as well as the router block. Noticed that the RCA is a heterogeneous array. The ALU supports up to 26 different operators, but only the ALUs in the second line contain 16x16 fixed-point multipliers and this architecture reduced up to 80% area comparing to a homogeneous one [16].

In [16], a performance comparison with two commercial processors, e.g. Intel Atom 230 (45 *nm*, 1600 *MHz* and 4000 *mW*) and ARM Cortex A8 (28 *nm*, 1000 *MHz*, and 300 *mW*) is made. REMUS is processed at 65 *nm*, working at 200 *MHz* with an average power consumption of 100 *mW*. Fig. 4 is the performance comparison based on 13 dwarfs [17]. REMUS outperforms the general purpose processors of 2.5x and 40.x on average, respectively. And the power consumption is only 1/40 and 1/3, respectively.

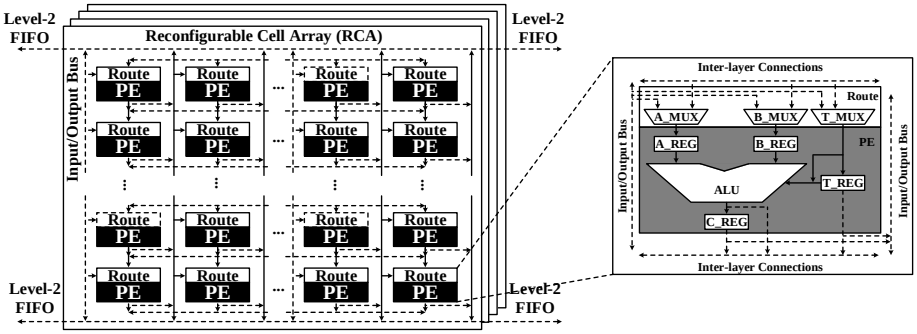

Fig. 3: The architecture of REMUS PEA and PE.

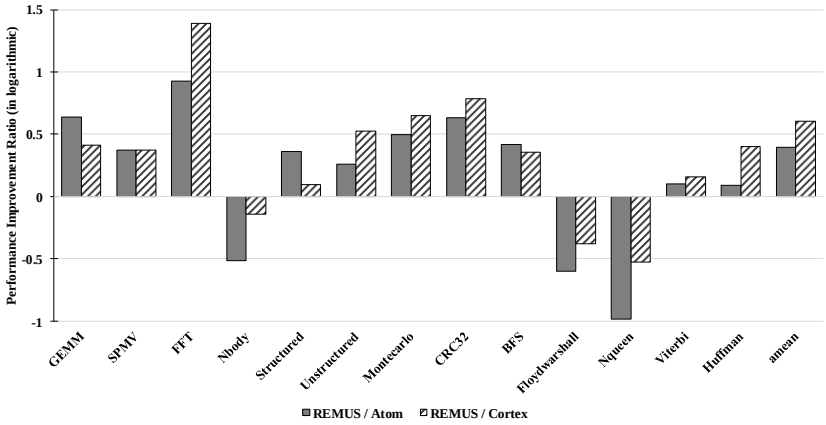

Fig. 4: Performance improvement ratio of REMUS comparing with Intel Atom 230 and ARM Cortex A8 [16].

2.2. Advantages of spatial computation

2.2.1. Flexibility of computation form

Ignoring the memory architecture of the PE array in Fig. 1, the connecting network of the array offers a direct point-to-point communication channel for the PE. An obvious benefit of this channel is that it alleviates the pressure of the memory operation dependence by transmitting the intermediate results to a successive instruction and not involving the memory access or even register file. This strategy both increase the throughput and lower the memory accessing conflict. And as shown in Fig. 2(d), the spatial architecture can be organized as a flexible pipeline to take the advantage of coarse-grained pipeline parallelism.

2.2.2. Massive computation resources

Comparing with conventional cores which apply deep pipelines and other complex technologies to guarantee acceleration, spatial computing architecture uses lightweight cores. In the conventional cores, each core only contains a small part of components directly related to the execution of operations. A bunch of units inside the chip are used to handle the auxiliary workloads, e.g. in superscalar processor, grouping the instructions and data. Contrarily, spatial architecture's die area is mainly for operations and the communication between the neighboring execution units. Consequently, the computation resources density is more intensive than conventional ones, such as MT. Monsoon [18] and Wavescalar [10] are both containing over 1000 PEs or function units (FUs) [13].

2.2.3. Scalability

The core-to-core communication model enables the spatial computation extremely high locality. In some implementations, each execution unit only connects and communicates with its spatial neighbors, totally aborting the centralized communication such as shared memory or full interconnections. The locality enables

each core responsible for the neighbors only, without the unconnected cores. Since the topology of interconnection is scalable in two or three dimensions, the computation array can be scaled as large as the application required.

2.2.4. Energy efficiency

According to [4] [8], the total power dissipation of a processor can be formulated as follow:

$$P_{total} = P_{dynamic} + P_{static} = n \cdot \alpha \cdot C \cdot V_{DD}^2 \cdot f + n \cdot V_{DD} \cdot I_{leakage} \cdot k_{design} \quad (2)$$

As the transistor size scaling down, the connection wires did not follow the trends. The conventional processors suffered severely from the increasing of the average load capacitance, and leakage current introduced by poor scaling down of the communication wires. However, spatial computing architecture applying the lightweight core configuration and point-to-point communication strategy alleviates both the inter-core and intra-core communication wires scaling down problems. Secondly, to keep the locality and decentralization, spatial computing architecture puts caches or buffers more close to the cores, and even the on-chip memory is separated to serval part as a memory bank attached to each individual core [7]. Short distance does not only mean short memory accessing delay, but also low power consumption in both data and instruction transmission as well as a lower memory accessing conflict. It is worth noticing, that the instructions in spatial architecture spreading in space as shown in Fig. 2(d), also saves the energy of fetching and decoding the instructions like shown in Fig. 2(b).

2.3. The Challenges for Spatial Computation

Though many benefits can be obtained by applying the spatial computing architecture, there are still some challenges which constrain its suitable application scenarios.

2.3.1. Programmability and compatibility issues

Coding in the spatial architecture appears to be more difficult than in the conventional way. Most popular high level programming languages are not originally designed for the spatial architecture. Though, there are some variants of programming languages attempt to support it and aiming to enable the isolation of programming and hardware detail. But they're still far away from the required programmability. On the other hand, the spatial programming needs the programmers to extract the parallelism from the application and expose them to the high level abstraction of the hardware [4], meeting the demands of various real world applications. Current spatial languages are more like writing code to describe both the application and hardware, and the desired high level language is that it can help to describe the application requirements, of course under the circumstance of spatial architecture. They may sound similar, but there is a huge difference between them that the former required programmer to consider two problems in one description while the latter only needs one.

2.3.2. Trade-offs between global and local

In the section about the advantages of spatial architecture, the locality offers the processor more scalability and more intensive computation density. But when it comes to programming, the locality sometimes troubles the programmer or compiler. They have to carefully arrange the distribution of instructions and data. In some situation, it is even no possible to find an optimal solution for the placement. Sometimes the programmer or compiler have to utilize the global resources, e.g. shared memory or complex interconnection network, to mitigate the divergence.

2.3.3. Trade-offs between heavy or light

Conventional processors usually employ some dedicated instruction management resources to support dynamic extracting concurrency, handling data dependency and control dependency, e.g. register renaming, out-of-order execution and re-order committing, branch prediction, speculative execution, etc. All of these increase the complexity of the processor and decrease the energy efficiency in other aspect. A lightweight core is unable to carry all these functions. A powerful static scheduling strategy can help, but is limited. A dynamic scheduling requires the runtime information and makes it impossible to solve all the problem statically during the compilation time.

3. Conclusion

As the coming of many-core era, spatial computing architecture brings some new idea to meet the requirement of various applications from different scenarios, ranging from signal processing to high-performance computing. These scenarios sharing a characteristic that they have data level parallelism, less control divergence. For some control intensive tasks, the spatial architecture is overwhelmed by some conventional processors, e.g. out-of-order superscalar processor. These challenges mentioned above can be mitigated by some specific solutions, e.g. a hybrid architecture of spatial computation and traditional processor. Though these efforts, spatial computation has gradually become a part of the mainstream computing infrastructure.

4. Acknowledgements

This work is supported by the National High Technology Research and Development Program of China (Grant No. 2012AA012701) and National Natural Science Foundation of China (Grant No. 61672317).

5. References

[1] D. A., Patterson. Computer architecture: a quantitative approach. *Elsevier*, 2011.

[2] D. A., Patterson, and L. H., John. Computer organization and design: the hardware/software interface. *Newnes*, 2013.

[3] W. A., Wulf, and S. A., McKee. Hitting the memory wall: implications of the obvious. *ACM SIGARCH computer architecture news* 23.1 (1995): 20-24.

[4] A. M., Zaidi. Accelerating control-flow intensive code in spatial hardware. *No. UCAM-CL-TR-870*. University of Cambridge, Computer Laboratory, 2015.

[5] G. M., Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM, 1967.

[6] M., Budiu, et al. Spatial computation. *ACM SIGARCH Computer Architecture News*. Vol. 32. No. 5. ACM, 2004.

[7] M., Pellauer, et al. Efficient Control and Communication Paradigms for Coarse-Grained Spatial Architectures. *ACM Transactions on Computer Systems (TOCS)* 33.3 (2015): 10.

[8] J. A., Butts, and G. S., Sohi. A static power model for architects. *Microarchitecture, 2000. MICRO-33. Proceedings. 33rd Annual IEEE/ACM International Symposium on*. IEEE, 2000.

[9] V., Govindaraju, C. H., Ho, and K., Sankaralingam. Dynamically specialized datapaths for energy efficient computing. *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on. IEEE*, 2011.

[10] S., Swanson, et al. The wavescalar architecture. *ACM Transactions on Computer Systems (TOCS)* 25.2 (2007): 4.

[11] D., Burger, et al. Scaling to the End of Silicon with EDGE Architectures. *Computer* 37.7 (2004): 44-55.

[12] K., Sankaralingam, et al. TRIPS: A polymorphous architecture for exploiting ILP, TLP, and DLP. *ACM Transactions on Architecture and Code Optimization (TACO)* 1.1 (2004): 62-93.

[13] F., Yazdanpanah, et al. Hybrid dataflow/von-Neumann architectures. *IEEE Transactions on Parallel and Distributed Systems* 25.6 (2014): 1489-1509.

[14] C., Kim, et al. Composable lightweight processors. *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on.* IEEE, 2007.

[15] B., Robatmili, et al. Strategies for mapping dataflow blocks to distributed hardware. *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on. IEEE*, 2008.

[16] L., Liu, et al. An energy-efficient coarse-grained reconfigurable processing unit for multiple-standard video decoding. *IEEE Transactions on Multimedia* 17.10 (2015): 1706-1720.

[17] K., Asanovic, et al. The landscape of parallel computing research: A view from berkeley. Vol. 2. *Technical Report UCB/EECS-2006-183*, EECS Department, University of California, Berkeley, 2006.

[18] G.M., Papadopoulos, and K. R., Traub. Multithreading: A revisionist view of dataflow architectures. *ACM SIGARCH Computer Architecture News*. Vol. 19. No. 3. ACM, 1991.