DTP: Enabling Exhaustive Exploration of FPGA Temporal Partitions for Streaming HPC Applications

Mostafa Koraei
School of Electrical and Computer Engineering
University of Tehran
Tehran, Iran
m.koraei@ut.ac.ir

Magnus Jahre
Dept. of Computer and Information Science (IDI)
NTNU NO-7491
Trondheim, Norway
magnus.jahre@idi.ntnu.no

S.Omid Fatemi
School of Electrical and Computer Engineering
University of Tehran
Tehran, Iran
omid@fatemi.net

ABSTRACT

Reconfigurable computing systems show great promise for accelerating streaming HPC applications because of their low power consumption and high performance. However, mapping an HPC application to a reconfigurable system is a challenging task. The challenge is exacerbated by the need to temporally partition computational kernels when application requirements exceed resource availability. In this paper, we propose a novel design methodology that we call *Data-flow Temporal Partitioning (DTP)*. The key insight in the design of DTP was that the application should be represented as a high-level data flow graph where each node is a computational kernel and the edges represent inter-node data flow. DTP also supports parallel instantiation of kernels and multiple kernel implementations at different performance/area design points. In contrast to previous proposals, DTP is able to exhaustively explore the solution space for practical applications. Our evaluation of DTP shows that it is able to identify candidate implementations that outperform both previously proposed partitioning heuristics and a direct mapping to the synthesis tool. The temporal configuration selected by DTP can outperform the direct mapping by up to 3X.

Keywords

High performance computing, Field programmable gate array, Data flow graph, Temporal partitioning, Design space exploration.

1. INTRODUCTION

High Performance Computing (HPC) is characterized by the desire to achieve maximum performance for important applications from diverse fields such as climate modelling, car and aircraft design, energy, material science and drug design. HPC applications are usually data intensive and parallel and may run for days and possibly weeks. Furthermore, current and future HPC systems are expected to be power limited [15]. This, and the Dark Silicon effect [4], have motivated researchers to explore accelerating HPC applications on non-conventional architectures [17, 16]. One promising strategy is to leverage emerging reconfigurable devices to develop *High Performance Reconfigurable Computers (HPRCs)*. HPRCs consist of a conventional host microprocessor and one or more *Reconfigurable Processing Units (RPUs)* that can be one or multiple FPGAs.

Mapping an HPC application onto an RPU is a challenging and time-consuming task. This process can be simplified by adopting a high-level representation of the application. On possibility is to represent the application as a *Directed Acyclic Graph (DAG)* where the nodes represent computational kernels and the edges represent data flow. This representation is especially well suited to HPC applications where data is streamed through pipelined computations. HPRC development is further complicated by the observation that an HPC application often requires more resources than the RPU can provide. Therefore, many researchers [8, 18, 7] have tried to exploit *Temporal Partitioning (TP)*. With TP, the DAG is partitioning into temporally interconnected sub-graphs and each sub-graph becomes an FPGA configuration. Unfortunately, applying TP results in a large design space. It is not feasible for a designer to explore this design space manually, and there is a need for automated *Design Space Exploration (DSE)* methodologies.

Existing TP techniques represent the computational kernels at a low level of abstraction [20, 19, 2]. This choice significantly increases the size of the design space since it grows exponentially with the number of nodes in the graph. Therefore, researchers have proposed heuristics [12, 14] such as minimum number of FPGA configurations [21], minimum communication cost [18, 12] and maximum resource utilization [2] as a practical means to finding acceptable temporal partitions. However, these heuristics select partitions based on local criteria which may not be a good for the complete application.

In this work, we propose a novel design space exploration methodology which we call *Data-flow Temporal Partitioning (DTP)*. The key observation that led us to propose DTP was that the computational kernels should be represented at a high abstraction level. This limits the size of the design space, and results in the DFGs of a real world HPC applications such as SPH pressure force calculation [12], DAB receivers [5], 2D FDTD updates [9] and JPEG encoding [1] containing maximum 20 or 30 nodes. The key advantage is that DTP is able to exhaustively explore the design space.

However, raising the abstraction level may also potentially hide interesting configurations. The designer can alleviate this problem by applying DTP in an iterative fashion. Because we use commercial HLS tools, defining coarse grained processing nodes leads to more accurate timing and area estimation for a configuration than using many fine grained PN implementation information for estimation because of accumulating errors of many small PNs is much more than a few large PNs. This approach finally reduces the errors of execution time estimation of the whole application.

To evaluate DTP, we have generated 15 synthetic SDFGs with between 10 and 12 processing nodes. We used Vivado HLS to generate three different implementations of each node to mimic an implementation library. Our experiments show that DTP improves execution time by up to 3X compared to a baseline containing a single temporal partition. We also found that the optimal partitioning depends on the graph topology, processing node resource requirements, the throughput of each node, the available hardware resources, the size of the problem and the available bandwidth. Consequently, one cannot expect to find a near-optimal partitioning by using the simple heuristics proposed in prior work.

2. DATA-FLOW TEMPORAL PARTITIONING (DTP)

Figure 1 illustrates the detailed sub modules of DTP. The letters A, B and C in Figure 1 refers to the three main steps of DTP that are application-, hardware- and problem size related, respectively. Step A determines all possible partitionings and their base configurations of the Input SDFG. In step B, DTP prunes the design space by taking into account the constraints of the hardware target. Finally, DTP determines the performance of each partitioning in step C and presents feasible solutions to the designer. For convenience, Table 1 contains a list of the abbreviations we use when discussing DTP.

2.1 Application Model

DTP assumes that the application is represented as a Synchronous Data Flow Graph (SDFG). We assume that all application control flow can be determined at compile time. This assumption is reasonable for many scientific and high performance computing applications such as reverse time migration [6] or N-Body simulations [10]. These applications essentially stream massive amounts of data through a computational pipeline, and the application control flow is given by the path taken by each data element.

In this paper, we assume that the designer defines a *Processing Node (PN)* as a small, meaningful portion of the application. For instance, the designer can consider computational kernels of the application such as spatial convolution, FFT, noise cancellation filters as PNs. For each processing node, we require that the designer provides the following inputs to DTP: *Produced Packet Per Iteration (PPI)*, *Required Packet Per Iteration (RP)* from each input edge, *Packet Size (PS)* and *Total number of produced packets (TP)*.

An *Application Edge (AE)* connects two PNs together. Since each PN produces only one type of data packet, all application edges that have a common source node must also carry same data type. Therefore, if an FPGA configuration is being feed by multiple AEs with one source node, in the

Table 1: Abbreviations

Description	Abbr
Input Model	
Processing Node	PN
Application Edge	AE
Memory Node	MN
PN Produced Packet Per Iteration	PPI
Packet Per Iteration	PI
PN Required Packet Per Iteration	RP
PN Total number of Produced Packets	TP
Edge Packet Size	PS
Source Node of and edge	ScN
Sink Node of an edge	SnN
Throughput of a PNI	NIT
Implementation	
Processing Node Implementation	NI
clock frequency of a PNI	NI_C
Iteration interval of a PNI	NI_I
PNI required hardware, item: LUT, FF, ...	NI_{item}
Methodology	
Configuration	C
Partitioning	P
Number of configurations of a partitioning	P_{NC}
Processing Node of a Configuration	CPN
Configuration of a Partitioning	PC
Configuration Implementation	CI
Configuration Implementation clock frequency	CIC
Configuration Implementation Instance Throughput	CIT
number of instances of a Configuration Implementation	N_{CI}
Configuration total output bytes	C_{TO}
Configuration total input bytes	C_{TI}
Configuration Input Bytes Per Cycle	C_{IB}
Configuration Output Bytes Per Cycle	C_{OB}
Execution time of a CI	T_{CI}
Execution time of a partitioning	T_{EP}
Configuration Implementation required LUT	CI_L
Configuration Implementation required FF	CI_F
Configuration Implementation required DSP	CI_D
Configuration Implementation required BRAM	CI_R
Configuration Best Execution Time	T_{CB}
Configuration Output Packet Size	C_{PS}
Hardware Resources	
Available FF	H_F
Available LUT	H_L
Available DSP	H_D
Available BRAM	H_R
Host to FPGA Bandwidth	BW_I
FPGA to Host memory Bandwith	BW_O

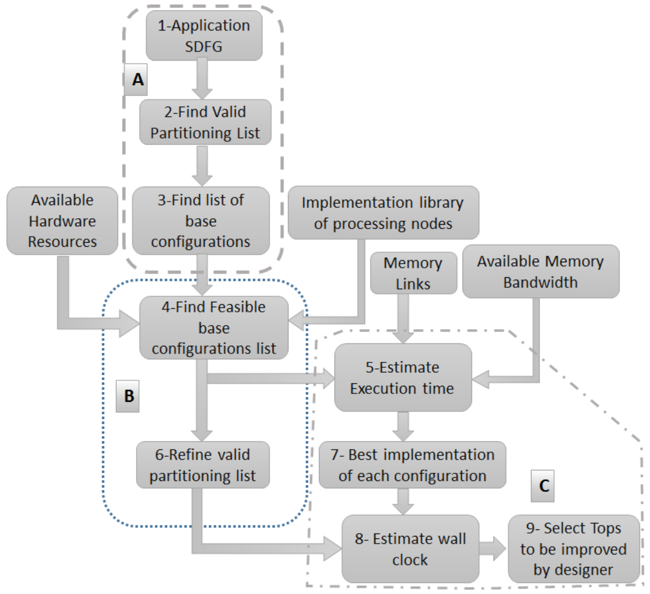
1-Application SDFG
A
2-Find Valid Partitioning List
Available Hardware Resources
3-Find list of base configurations
Implementation library of processing nodes
Memory Links
Available Memory Bandwidth
4-Find Feasible base configurations list
5-Estimate Execution time
B
6-Refine valid partitioning list
7- Best implementation of each configuration
C
8- Estimate wall clock
9- Select Tops to be improved by designer

Figure 1: Overview of DTP methodology

performance calculations DTP considers one edge for them with the maximum sink node data rate. Outgoing edges from one configuration follows same rule.

2.2 Processing Node Implementation Library

DTP requires that the designer provides a library of different processing node implementations. The designer can decide to use low level HDL or HLS tools to implement each PN. For a specific PN, the implementation library has three classes of properties. The first class of properties are the timing properties that consist of clock frequency, latency, iteration interval and throughput. Second, communication properties are the required input and output bandwidth for each implementation instance. Finally, the resource requirement properties are the number of LUTs, number of flip flops, DSP modules, BRAMs and SLR registers.

2.3 Configuration and Partitioning Objects

DTP uses the application model to produces two objects that we refer to as the configuration and the partitioning. The configuration is a sub-graph of the application SDFG with all its interior edges. This sub-graph will is a candidate to be a temporal configuration implemented in the FPGA. A configuration is *valid* if there is not any directed path between any pair of its nodes that passes from a node that are not part of the current configuration. In other words, a valid configuration is a part of the application that can be independently synthesized such that it carries out meaningful work for the application. A *feasible* configuration is a valid configuration where the resource requirements can be satisfied by the target FPGA.

We refer to a set of valid configurations which covers all PNs of the SDFG and where each node is located in only one configuration as a partitioning. A valid partitioning is a partitioning that for each node of its configurations all of the predecessor nodes have been located in previous configurations. In other words, a partitioning is valid when all

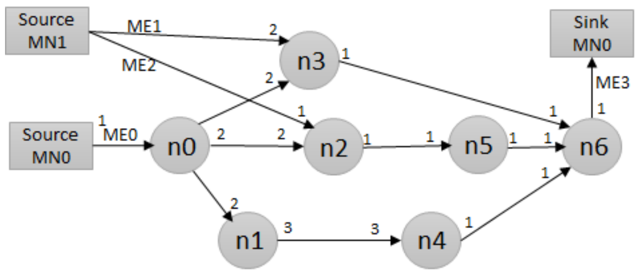

Figure 2: SDFG Example

input data needed for a particular configuration has been produced by previous configurations.

2.4 The Main Steps of DTP

Figure 1 illustrates the different steps of the DTP methodology. DTP takes the application model as input and finds all possible valid partitionings and their base configurations in *Step A*. The first task in Step A is to make the SDFG model of application. Here, we prepare a table of application edges properties that contains one row for each edge. Then, DTP finds all possible valid partitionings by first categorizing nodes in dependency based groups before it finds a path from the start node to one of the nodes in the last group. We refer to this path as the critical path of the SDFG, and it is not necessarily unique.

Figure 2 contains the SDFG of an application that is our acceleration target. By applying dependency-based grouping on this graph we find that group 0 contains node [n0], group 1 contains nodes [n1,n2,n3], group 2 contains nodes [n4,n5] and group 3 contains node [n6]. The critical path of the SDFG can be [n0, n2, n5, n6] or [n0, n1, n4, n6], and it does not matter which one is selected. For this example, we select path [n0, n2, n5, n6]. After finding the critical path of the input SDFG, DTP uses it as a simple straight directed graph and makes a list of all its dependent valid partitionings. To find the valid partitionings of the complete graph, DTP adds nodes one by one in grouped order to this list.

An addition of node to a configuration of a partitioning is valid if following conditions are true. First, all predecessors of the node should be appeared in the current or previous configurations. Second, any directed path between current node and other nodes of the current configuration should be covered by the current configuration. After finding all valid partitionings, DTP fills the list of base configurations. In the SDFG of Figure 2, [n0,n2,n3], [n1,n3] and [n1,n2,n4.n5] are examples of base configurations.

In step B, DTP finds valid partitionings by considering the hardware constraints. Task 4 uses the implementation library and information provided regarding available hardware resources to evaluate the feasibility of the items in the implementation list of each configuration.

$$\forall items \in H_{items} \sum_{n \in CPNs} NI_{n_{item}} \leq H.item \qquad (1)$$

Equation 1 formulates the feasibility criteria of a CI. Here, *item* is one of hardware resource parameters (i.e, LUT, DSP, FF or BRAM). DTP removes the configurations that do not have any feasible implementations from the list of base configurations. In Step 5, DTP removes all partitionings that contain completely removed base configurations from the partitioning list. In the SDFG of Figure 2 the base configuration [n2,n3,n5] is not feasible if the target hardware has 600 DSP modules and n2, n3 and n5 require 300, 120 and 270 DSP modules, respectively. Thus, we will not have any partitioning after Step B that contains this configuration.

In Step C, DTP calculates the execution time of all feasible configurations and all valid configurations. Step 6 gets its data from fourth step and can be done in parallel with fifth step. In this step, DTP calculates the execution time of each feasible CI of each base configuration by using memory links and information about amount of data. The following formulas are used in Step 3 of DTP to estimate the execution time of CIs.

$$C_{TO} = \sum_{i \in CPNs, j \notin CPNs} PS_{i,j} * TP_{i,j} \quad (2)$$

$$C_{TI} = \sum_{i \notin CPNs, j \in CPNs} PS_{i,j} * TP_{i,j} \quad (3)$$

$$CIT = min(\frac{NI_C}{NI_I})_{\forall NI \in CI} \quad (4)$$

$$N_{CI} = min(\frac{H_L}{CI_L}, \frac{H_F}{CI_F}, \frac{H_D}{CI_D}, \frac{H_R}{CI_R}) \quad (5)$$

$$C_{IB} = \sum_{i \in C_{In\ Edges}} SnN_i.PI * PS_i \quad (6)$$

$$C_{OB} = \sum_{i \in C_{Out\ Edges}} ScN_i.PI * PS_i \quad (7)$$

$$T_{CB} = min(\frac{C_{TI}}{min(N_{CI} * CIT * C_{IB}, BW_I)}, \frac{C_{TO}}{min(N_{CI} * CIT * C_{OB}, BW_O)}) \quad (8)$$

In task 7, DTP selects the CI with the shortest execution time for each base configuration, and its execution time is considered as the configuration execution time. In the task 8, the wall clock time of each partitioning is calculated by adding the execution time of all its configurations as well as adding the total configuration time overhead. Finally, DTP ranks the best performing partitionings and presents a configurable number of them (e.g. 10) to the developer. These implementations can either be used directly or studied in order to find a better SDFG formulation.

For example, DTP may find that the best partitioning is combined of the configurations [n0,n2,n3], [n1,n5] and [n4,n6]. From the execution time estimates, the designer is made aware that execution time of the third configuration is than 50% higher than the average execution time of the other configurations. In this case, the designer has two options. First, he can focus on either improving the implementation of [n4,n6]. Second, he can reformulate n4 and n6 to use smaller PNs and then run DTP to find the best partitioning of this new SDFG.

3. EXPERIMENTAL METHOD

We prepared 20 PNs and used these to create 15 synthetic SDFG benchmarks with 10 to 12 PN each. Our set of benchmarks covers both long pipeline models of applications as well as applications with parallel nodes. The PNs in each benchmark were randomly selected from our set of 20 PNs, and each type of PN is used only once in a benchmark SDFG. For PNs, we used three different types of processing nodes: 3*3 matrix and stencil operations, polynomial calculations and triangular calculations. The PNs have a different number of inputs from 1 to 4 and take different packet sizes as inputs. Each PN has three different PNIs in the implementation library.

We used Xilinx Vivado to synthesize each PN with three different options: fast, normal and slow. This gives us a range of implementations with different performance and area requirements. We assume that the designer has four options for hardware resources of the target HPRC (i.e. the XC7V585T, XC7VX690T, XC7V2000T and two XC7V200Ts). The baseline for comparison is the execution time of the application when using a single configuration.

4. RESULTS

We calculated the execution time of each benchmark when its SDFG is implemented completely in one configuration on the XCV2000T. Then, we explored the temporal partitioning design space of each benchmark with three sets of NIs and find the best partitioning for each benchmark. Figure 3 shows the speed up of different sets of NIs and also the partitioning that selected with DTP compared with one time configuration of the FPGA. Series TPI1, TPI2 and TPI3 determines best speed up of temporal partitioning with leveraging node parallelism when each node can be implemented in only one type of implementation library. TPI1 means all node are implemented based on slow and area efficient set of NI and TPI3 means that all nodes are implemented based on fast and high resource required set of NIs.

We also explored the design space of TP with DTP when it leverages all three NI types. In many of explored benchmarks, TP lead in better performance than a one time implementation. As seen in Figure 3, the difference between the speed up gained by DTP over the best performing NI type is significant. For some benchmarks, such as SDFG8 and SDFG11, this difference is more than 25% compared to a direct implementation of the SDFG using Vivado. On average, DTP provides more than 20% speed-up. This is expected because DTP can combine NIs from different library sets to achieve a better execution time.

Figure 3 also shows that selecting a set of NIs with higher performance does not always result in lower execution time for the complete application. This is shown with SDFG3 and the second and third NI sets. In this benchmark, the best performance of partitioning with third NI set is lower than second one. At the same time, the performance of each implementation in third set is higher than the implementation of the same PN in the second set. The reason for this is that DTP uses data parallelism and each configuration may be implemented in several instances on the FPGA. This provides evidence that defining local criteria to find temporal partitions may result in sub-optimal application performance.

Figure 4 shows the number of FPGA configurations in the best selected partitioning for each set of NIs. As this

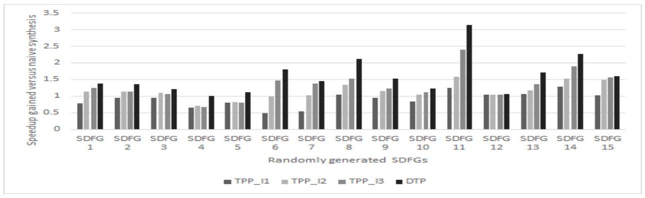

Figure 3: Speedup of selected partitioning by DTP vs one time configuration

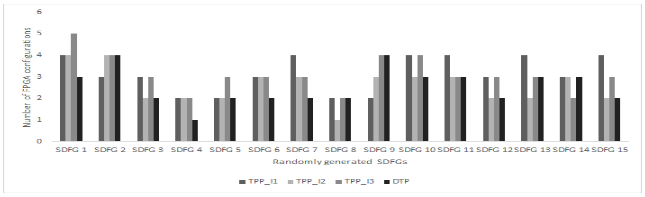

Figure 4: Number of configurations for the best partitioning

figure shows the performance of the applications is not directly related to the number of FPGA configurations. One of the reasons is that we assumed our target applications to be data intensive. The execution time of this type of applications is so long that the reconfiguration time of FPGAs is negligible. Figure 6 makes this concept more clear, for SDFG6 and SDFG15 the speedup of two times configuration is more than best partitioning with 4 configurations; but for SDFG11 and SDFG13 the speed up of best partitioning with 3 and 4 configurations is significantly more than best partitioning with 2 configurations. Since selecting the number of configurations as the criteria of temporal partitioning can mislead the designer.

Figure 5 illustrates the change in the execution time of 5 benchmarks with 4 different hardware resources. In this experiment, we changed the available hardware resources and found the best execution time of each benchmark. This figure shows that DTP can help the designer to make an informed trade-off between performance and hardware overhead.

5. RELATED WORK

Many researchers tried to gain higher performance of HPRC by temporal partitioning. The bandwidth of an FPGA to external memory is an intrinsic bottleneck. Therefore some researchers tried to minimize the inter configuration communication cost to achieve better performance. One algorithm to make partitioning with minimum interconnection under area constraints is proposed in [13]. Authors tried use network flow-based algorithms for initial partitioning of the graph to obtain a cut set and reduce maximum communication cost. Reduced Data Movement Scheduling (RDMS) in [12] was proposed to reduce the inter-configuration communication overhead and FPGA configuration overhead. Although RDMS exploits a library of implementations but its criteria in refining partitioning is the cost of inter- configuration communication. In [19] a Pareto optimal temporal partition methodology based on multi-objective GA proposed

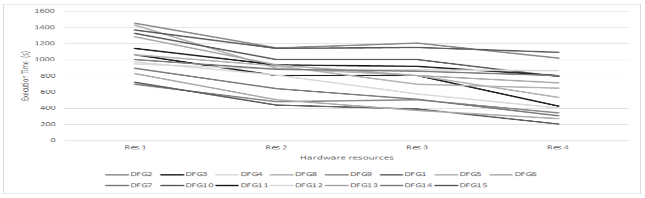

Figure 5: Execution time of the best selected partitioning for different hardware resource

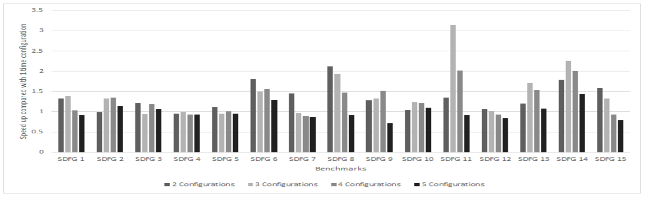

Figure 6: Best speed up achieved on different number of FPGA configurations

by authors, the aim of the paper was reducing the communication cost. This methodology can explore a big design space compare to DTP, but because of its criteria there is no guarantee to find the best partitioning.

Another issue in HPRC systems is the time of reconfiguration. Some researchers tried to use the device area as much as possible to reduce the number of configurations. In [10] authors proposed an approach for task mapping in RC hardware based on GA that exploits different implementations of tasks. They used variant implementations but the difference of their approach to DTP is that tried to use maximum utilization of the device so that results in each configuration had has only one instance in the FPGA. In [5] authors proposes an algorithm for TP that yields a chain of configurations which are balanced in terms of resources, while jointly have a minimal data throughput but their approach did not consider data dependency in a DFG and only works for straight pipeline chains.

Most close works to this paper have been done by researchers that tried to explore the design space by heuristic methods for different implementations of each PN. In [3] authors used a library of implementation of tasks and propose a DSE methodology for TP based on Tabu search strategy. The criteria of selection was not directly based on the total application performance since in some cases this algorithm lost many of fast partitionings. In [11] a genetic algorithm based methodology proposed for hardware task mapping that used a library of fixed implementations for each task. In [14] author proposed a performance oriented TP approach that exploits data penalization on RC devices with the assumption that there unlimited resources. In this research the variant in implementations was not considered and because of no constraint on resources task duplication is possible with GA based proposed method.

As mentioned in previous related works DTP is the first methodology that consider different implementations of processing nodes, different problem size, multi-instances of an FPGA configuration, arbitrary application DFG and focuses on execution time as the selecting criteria.

6. CONCLUSION

In this work, we proposed the Dataflow Temporal Partitioning (DTP) design methodology. The objective of DTP is to efficiently map data intensive HPC applications onto reconfigurable platforms while leveraging temporal partitioning. DTP consists of 3 independent steps that collectively help the designer to explore the temporal partitioning design space more efficiently. DTP considers the performance effects of problem size and a range of technology-dependent implementation characteristics. We compared the results of DTP with results of a standard synthesis tool and found that the speed up in some cases is significant.

7. REFERENCES

[1] S. Banerjee, E. Bozorgzadeh, and N. Dutt. Exploiting application data-parallelism on dynamically reconfigurable architectures: Placement and architectural considerations. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(2):234–247, 2009.

[2] Z. Chen, M. Qiu, Z. Ming, L. T. Yang, and Y. Zhu. Clustering scheduling for hardware tasks in reconfigurable computing systems. *Journal of Systems Architecture*, 59(10 PART D):1424–1432, 2013.

[3] P. S. B. do Nascimento, V. W. de Medeiros, V. L. Souza, A. C. Barros, and M. E. de Lima. A Temporal Partitioning Methodology for Reconfigurable High Performance Computers. *2008 International Conference on Reconfigurable Computing and FPGAs*, pages 307–312, 2008.

[4] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark Silicon and the end of Multicore Scaling. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 365–376. IEEE, 2011.

[5] M. Feilen, A. Iliopoulos, M. Vonbun, and W. Stechele. Weighted partitioning of sequential processing chains for dynamically reconfigurable fpgas. *2013 23rd International Conference on Field programmable Logic and Applications*, pages 1–8, Sept 2013.

[6] H. Fu, L. Gan, R. G. Clapp, H. Ruan, O. Pell, O. Mencer, M. Flynn, X. Huang, and G. Yang. Scaling reverse time migration performance through reconfigurable dataflow engines. *IEEE Micro*, 34(1):30–40, 2014.

[7] W. Fu and K. Compton. An execution environment for reconfigurable computing. *13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05)*, pages 149–158, April 2005.

[8] K. M. Gajjalapurna and D. Bhatia. Partitioning in Time : A Paradigm for Recon gurable Computing. *International Conference on Computer Design: VLSI in Computers and Processors*, pages 340–345, 1998.

[9] H. Giefers, C. Plessl, and J. Förstner. Accelerating finite difference time domain simulations with reconfigurable dataflow computers. *SIGARCH Comput. Archit. News*, 41(5):65–70, June 2014.

[10] M. Huang, V. K. Narayana, M. Bakhouya, J. Gaber, and T. El-Ghazawi. Efficient mapping of task graphs onto reconfigurable hardware using architectural variants. *IEEE Transactions on Computers*, 61(9):1354–1360, Sept 2012.

[11] M. Huang, V. K. Narayana, and T. El-Ghazawi. Efficient mapping of hardware tasks on reconfigurable computers using libraries of architecture variants. *Proceedings - IEEE Symposium on Field Programmable Custom Computing Machines, FCCM 2009*, pages 247–250, 2009.

[12] M. Huang, V. K. Narayana, H. Simmler, O. Serres, and T. El-Ghazawi. Reconfiguration and communication-aware task scheduling for high-performance reconfigurable computing. *ACM Trans. Reconfigurable Technol. Syst.*, 3(4):20:1–20:25, Nov. 2010.

[13] Y. C. Jiang and J. F. Wang. Temporal partitioning data flow graphs for dynamically reconfigurable computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(12):1351–1361, Dec 2007.

[14] C.-C. C. Kao. Performance-oriented partitioning for task scheduling of parallel reconfigurable architectures. *IEEE Transactions on Parallel and Distributed Systems*, 26(3):858–867, 2015.

[15] J.-F. Lavignon, D. Lecomber, I. Phillips, F. Subirada, F. Bodin, J. Gonnord, S. Bassini, G. Tecchiolli, G. Lonsdale, A. Pflieger, B. Andretti, T. Lippert, A. Bode, H. Falter, P. Blouet, and M. Muggeridge. ETP4HPC Strategic Research Agenda – Achieving HPC Leadership in Europe. Technical report, European Technology Platform for High Performance Computing, 2013.

[16] M. V. Lyashov, A. N. Bereza, J. V. Alekseenko, and L. M. L. Blanco. The hybrid reconfigurable system for high-performance computing. *Application of Information and Communication Technologies (AICT), 2015 9th International Conference on*, 2015.

[17] A. Mahram and M. C. Herbordt. NCBI BLASTP on High-Performance Reconfigurable Computing Systems. *ACM Transactions on Reconfigurable Technology and Systems*, 7(4):1–20, 2015.

[18] W.-K. Mak and E. F. Y. Young. Temporal logic replication for dynamically reconfigurable fpga partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(7):952–959, July 2003.

[19] W. Sheng, W. He, J. Jiang, and Z. Mao. Pareto optimal temporal partition methodology for reconfigurable architectures based on multi-objective genetic algorithm. *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 425–430, May 2012.

[20] C. Steiger, H. Walder, and M. Platzner. Heuristics for Online Scheduling Real-Time Tasks to Partially Reconfigurable Devices. *Proceedings of the 13th International Conference on Field Programmable Logic and Applications*, (2100):575–584, 2003.

[21] C. Yin, S. Yin, L. Liu, and S. Wei. Temporal partitioning algorithm for a coarse-grained reconfigurable computing architecture. *Proceedings of the 2009 12th International Symposium on Integrated Circuits*, pages 659–662, Dec 2009.