A SCALABLE DESIGN APPROACH FOR STENCIL COMPUTATION ON RECONFIGURABLE CLUSTERS

Xinyu Niu, Jose G. F. Coutinho and Wayne Luk

Dept. of Computing, School of Engineering, Imperial College London, UK
Email: {nx210, jgfc, wl}@doc.ic.ac.uk

ABSTRACT

Stencil-based algorithms are known to be computationally intensive and used in many scientific applications. The scalability of stencil algorithms in large-scale clusters is limited by data dependency between distributed workload. This paper proposes a scalable communication model to schedule communication operations based on available resources and algorithm properties. Experimental results from the Maxeler MPC-C500 computing system with four Virtex-6 SX475T FPGAs demonstrate linear speedup.

1. INTRODUCTION

The last few years have given rise to system architectures that are capable of significant performance, exhibiting larger number of processing cores and incorporating more hardware accelerators, such as GPUs and FPGAs. Efficient acceleration of stencil applications using FPGAs involves exploiting not only fine-grained parallelism, but also scaling applications to multiple FPGA nodes. Building reconfigurable stencil applications that scale linearly within a set of FPGA resources requires significant effort and high-level expertise.

In this paper, we address this design challenge with a scalable communication model which allows stencil-based applications, known to be both computationally intensive and difficult to parallelise, to explore available resources in reconfigurable clusters. The proposed approach presented in this paper focuses on the following challenges: how to utilise available resources in each FPGA, and how to ensure linear scalability when multiple FPGAs are involved. This work proposes a compilation approach that exploits properties of stencil computations in deriving data decomposition and communication scheduling strategies to efficiently adapt to implementations for multiple FPGAs. An option pricing application is developed with the proposed approach, demonstrating linear speedup with increasing resources.

2. RELATED WORK

Stencil computations have been extensively studied across various underlying platforms including many-core processors, hardware accelerators such as GPUs and FPGAs, and large-scale clusters. In 1984, the requirement to exchange boundary data between computing nodes was studied [1], and a subroutine call was proposed to handle this situation. The impact of domain decomposition on multi-processor systems was studied in 1987 [2], and the relationship between stencil halo size and communication requirements in grid environments was investigated in 2001 [3]. Recently, design parallelisation [4] and code generation [5] approaches have been proposed for many-core CPUs. In large-scale CPU clusters, communication patterns of stencil computation were optimised to provide scalable implementations [6, 7]. To exploit parallelism in GPUs, a reuse method of data stored in local memories was proposed to parallelise stencil designs inside GPUs [8], and asynchronous communication patterns were introduced to overlap the communication time with computation in multi-GPU systems [9]. Meanwhile, various stencil implementations have been proposed based on reconfigurable architectures. A streaming architecture was built to connect output data to the input of the following node [10]. However the architecture is limited to scenarios where domain decomposition is not required, i.e., stencil data size is small enough to fit into on-chip memory. A more general reconfigurable architecture was proposed to provide high performance of large-scale stencil computation [11], where results of multi-FPGA designs are extrapolated from single-chip designs. In this work, we aim at a design approach to generate scalable reconfigurable designs for large-scale stencil applications.

3. METHODOLOGY

Stencil-based algorithms perform a sequence of sweeps over a spatial domain, executing nearest neighbouring computation in each dimension of the spatial domain. One example of stencil code is shown in Algorithm 1, where the domain contains dimension x with size nx. $source$ and $target$

978-1-4799-0004-6/13/$31.00

Algorithm 1 Demonstration of a stencil algorithm.

```
1: int a0=0.1; int a1=0.2; int a2=0.4;
2: for t ∈ 0 → nt do
3:     for x ∈ 0 → nx do
4:         target[x] ← source[x] * a0 + source[x+1] * a1 + source[x-1] * a2;
5:     end for
6:     swap(target, source);
7: end for
```

are respectively one-dimension input array and output array, and $a0, a1, a2$ are stencil coefficients. Developers mark the code with a pragma that will be mapped to hardware. This code, like the one presented in Algorithm 1 (lines 3–5), must match a code pattern with the following features:

1. **a single loop or a perfect nested loop.** A loop nest is classified as perfect if all statements across the loops are confined in the innermost loop body. Loop transformations such as loop fusion and loop distribution can convert any nested loop into perfect loops nests [12].

2. **deterministic data access patterns.** Coefficients of loop index expressions, such as nx, must be constant values, either determined at compile-time or at run-time.

3. **platform constraints.** The specific hardware platform may have constraints that the design needs to meet. For example, our current hardware system does not support more than 15 I/O channels, so the number of input and output arrays cannot be larger than 11 (4 channels used for inter-FPGA communication).

C stencil descriptions from users are transformed into scalable streaming designs. Challenges for this compilation process include ensuring maximum performance for each node, balancing workload and scheduling asynchronous communication operations. DFGs with format shown in Figure 1 enable direct generation of streaming architectures. Resource consumption when design parallelism increases can be properly estimated, ensuring maximum design parallelism for available resources. Moreover, algorithm details can be extracted from DFGs, which, along with the optimised designs, support cycle-accurate predictions of computation/communication operations within each node of the mapped design. Therefore the communication operations can be scheduled to overlap computation operations without violating data dependencies. An example stencil algorithm is shown in Algorithm 1, where computation is limited to one dimension. Figure 1 and Figure 2 respectively present the DFG and the design template for Algorithm 1. Through this section, this simplified algorithm is used to demonstrate the optimisation process at compile-time.

As shown in Figure 1, nodes in a DFG include offset nodes, arithmetic nodes, parameter nodes and input/output nodes. Arithmetic nodes represent the arithmetic operations in stencil applications. These arithmetic nodes are mapped

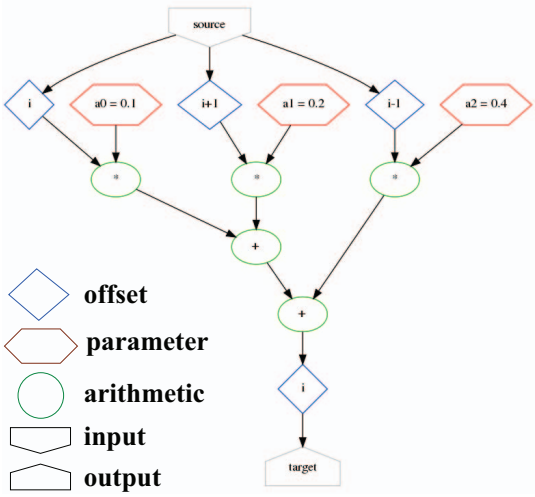

Fig. 1. Data flow graph extracted for Algorithm 1.

as a fully pipelined data-path. Provided required data, the data-path generates one result per clock cycle. The required data, on the other hand, are captured with offset nodes. Offset values correspond to the data position in arrays and the accessed clock cycles in hardware. The offset nodes are accumulated as on-chip memory architectures $\mathrm{mem_i}$ to buffer data accessed at different cycles, as shown in Figure 2. In the demonstration example, data accessed one cycle before and one cycle after are buffered as data streaming in.

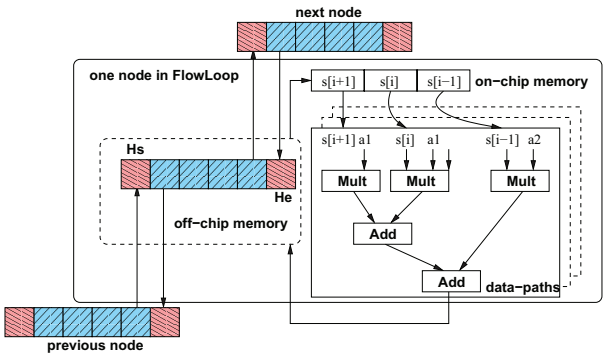

Fig. 2. Hardware architecture for stencil computation.

Data-paths are replicated to increase design parallelism. As fully pipelined data-paths cannot share resources with each other, LUTs resource consumption for data-paths can be accumulated as:

$$Ls = Par \cdot \sum_{i \in \odot} N_{\mathrm{i}} \cdot R_{\mathrm{L,i}} \quad \odot = \{+, -, \bullet, \div\} \qquad (1)$$

where Ls accounts the resource consumption for LUTs, N_{i} indicates the number of operators for arithmetic operation type i, R_{L} indicates the number of LUTs consumed, and

Par stands for design parallelism, i.e., the number of replicated data-paths. Similarly, FFs (Fs) and DSPs (Ds) resource consumption for data-paths can be estimated.

Instead of constructing an on-chip memory for each data-path, an optimised memory architecture can be shared by the replicated data-paths. As accessed data from the data-paths overlap with each other, the optimised memory architecture provides huge resource saving when data size increases. As an example, for Algorithm 1, increasing Par from 1 to 2 only expands on-chip data at cycle 1 from (0,1,2) to (0,1,2,3). The expanded on-chip memory is the union of data accessed by replicated data-paths. Its resource consumption is estimated as: $Bs = \bigcup_{i=1}^{Par} \mathrm{mem_i} \cdot R_\mathrm{B}$, where R_B is the memory resource consumption per data point.

To ensure peak performance of replicated data-paths, off-chip data are required to update the on-chip memory architecture. Increasing design parallelism linearly increases off-chip memory bandwidth. In the previous example, at cycle 2, data (2,3,4,5) are required by data-paths, two data are loaded from off-chip memory. Bandwidth requirements can be calculated as the difference between on-chip data at consecutive cycles.

$$Bw = \bigcup_{i=1}^{Par} \mathrm{mem_{i+1}} - \mathrm{mem_{i+1}} \cap \mathrm{mem_i} \tag{2}$$

Maximum Par can be achieved by comparing required resources and available local resources from infrastructure graph. In reconfigurable clusters, FPGAs with different sizes and capabilities need to cooperate with each other. The automatic optimisation process hides design details and heterogeneity in underlying hardware from users. The maximum Par, which can differ according the type of FPGA, is fed into the design template to derive efficient streaming architectures for all FPGA nodes in the cluster.

When a stencil design scales beyond single device, stencil domain is decomposed to balance workload with computation capacity, with data dependencies protected. As an example, if the x dimension is split into two sub-domains: $(0 \sim \frac{nx}{2})$ and $(\frac{nx}{2} + 1 \sim nx - 1)$, computing the boundary data $\frac{nx}{2}$ and $\frac{nx}{2} + 1$ requires data from other sub-domains. The required data are referred to as halo data. The halo data between two consecutive nodes are allocated into both nodes, and updated remotely during run-time, as shown in Figure 2. As halo data stand for boundary conditions at the beginning and ending of the allocated data, the halo region can be calculated by assuming offset data hit the boundary $mem_i = 0$:

$$\forall i \in mem_i = 0 : H_s = max_i \quad H_e = |min_i| \tag{3}$$

where H_s and H_e are the halo size at starting and ending area of decomposed domains, respectively. As each allocated node required $H_s + H_e$ additional data, the overall data size for a stencil design with N nodes is $ds + (N-1) \cdot (H_s + H_e)$. The allocated data D_n can be calculated as:

$$D_n = \frac{ds + (N-1) \cdot (H_s + H_e)}{\sum_{i=1}^{N} Par_i} \cdot Par_n \tag{4}$$

where design parallelism Par_n represents the computational capacity of node n, as each data-path generates one result per clock cycle.

Local communication constraints are built to protect data dependencies near the halo region. When a point d is processed, its neighbouring data $(d - H_s, d + He)$ must be within the same time iteration. Therefore, at each time iteration, the halo data calculated at remote nodes must flow into the local memory after halo data of current iteration have been streamed into on-chip memory, and before the halo data of next iteration are used. With design parameters and algorithms, local computation operations can be predicted cycle by cycle. Halo data H_s stream into on-chip memory at cycle 0 at the first time iteration and at cycle $\frac{D_n}{Par}$ at the second time iteration. The arrival time of starting halo data $T_{<s,n>}$ are bounded between these two time slots. Similarly, constraints for arrival time of ending halo data $T_{<e,n>}$ can be calculated.

$$0 \leq T_{<s,n>} < \frac{D_n}{Par_n} \tag{5}$$

$$\frac{D_n - H_e}{Par_n} \leq T_{<e,n>} < \frac{2 \cdot D_n - H_e}{Par_n} \tag{6}$$

A straightforward solution is to stall the computation when data are collected and redistributed. However, the communication overhead then increases with the loop size, which severely impacts the design scalability. To overlap communication time with computation time and ensure linear scalability, an asynchronous communication model is introduced. As shown in Figure 2, halo data H_e come from $(H_s, H_s + H_e)$ of next node. The generation time of H_e thus can be calculated as $\frac{H_s + H_e}{Par_{n+1}}$. If the remote halo data H_e are written into local memory once they arrive, the local constraints are violated. Local memory controllers insert delay d in the write command of halo data, updating halo data at a specific time slot. The arrival time of halo data thus is expressed as sum of the delay and remote generation time.

$$T_{<s,n>} = d_{<s,n>} + \frac{D_{n-1} - H_e}{P_{n-1}} \tag{7}$$

$$T_{<e,n>} = d_{<e,n>} + \frac{H_s + H_e}{P_{n+1}} \tag{8}$$

The inserted delay can be scheduled by comparing data arrival time with the boundary time slots.

4. RESULTS

A benchmark application, option pricing, is developed with the proposed approach. An option is a financial instrument

which provides its owner the right to buy or to sell an asset at a fixed price in the future. A `call option` allows owners to buy asset, while a `put option` allows owners to sell asset. Options are popular in the financial industry and pricing options usually involves solving partial differential equations (PDEs), especially the Black Scholes PDE [13]. The Black Scholes PDE with one variable (asset) following geometric Brownian motion is described as Eq.9, where $f_{t,s}$ denotes the price of the option, s denotes the value of the underlying asset, t denotes a particular time, τ is the risk-free interest rate, σ is the volatility of the underlying asset. Using explicit finite-difference expressions to replace the derivatives, the asset value $f_{t,s}$ can be calculated as in Eq.10, where α, β and γ are the constants determined by σ, τ and computational grid step size, constructing a 1-D stencil.

$$\frac{\partial f}{\partial t} + \tau s \frac{\partial f}{\partial s} + \frac{1}{2}\sigma^2 \frac{\partial^2 f}{\partial s^2} = \tau f \tag{9}$$

$$f_{t,s} = \alpha f_{t-1,s+1} + \beta f_{t-1,s} + \gamma f_{t-1,s-1} \tag{10}$$

The hardware designs are captured with MaxCompiler version 2012.1, implemented on Xilinx Virtex-6 SX475T FPGAs, each hosted by one of the four MAX3424A systems in an MPC-C500 computing node from Maxeler Technologies. As shown in Figure 3, using 48 parallel work units per FPGA, design performance increases linearly with available resources. When the stencil application is mapped into multiple FPGAs, the proposed approach ensures linear speedup for hardware implementations, achieving 487.8 GFLOPS for 4 FPGAs.

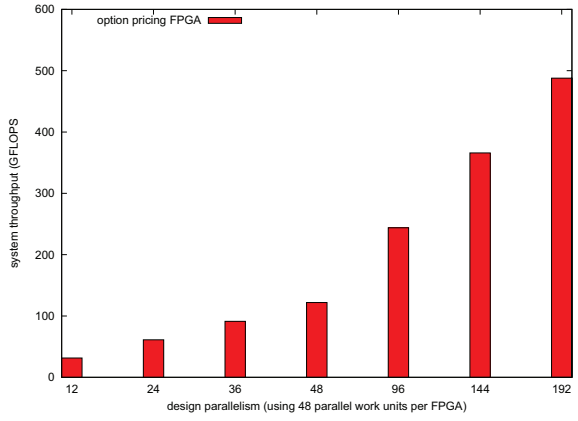

Fig. 3. System throughput of multi-chip implementations.

5. CONCLUSION

This paper presents a novel approach that automatically maps a C stencil computation, known to be computationally intensive and difficult to parallelise, into scalable hardware implementations. Our approach allows application developers to describe C stencil computations while abstracting from algorithm details. Experimental results show that the proposed approach achieves linear speedup when multiple FPGAs are involved in the target applications. Current and future work includes extending our approach to support a variety of applications and optimisation techniques, and to explore run-time improvement opportunities.

Acknowledgement

This work was supported in part by UK EPSRC, by the European Union Seventh Framework Programme under Grant agreement number 287804, 318521 and 257906, by the HiPEAC NoE, and by Xilinx.

6. REFERENCES

[1] G. C. Fox, "Concurrent processing for scientific calculations," in *COMPCON*, 1984, pp. 70–73.

[2] D. A. Reed, L. M. Adams, and M. L. Patrick, "Stencils and problem partitionings: Their influence on the performance of multiple processor systems," *IEEE Trans. Computers*, vol. 36, no. 7, pp. 845–858, 1987.

[3] M. Ripeanu, A. Iamnitchi, and I. T. Foster, "Cactus application: Performance predictions in grid environments," in *Euro-Par*, 2001, pp. 807–816.

[4] H.-J. Bungartz, A. Heinecke, D. Pflüger, and S. Schraufstetter, "Parallelizing a black-scholes solver based on finite elements and sparse grids," in *IPDPS Workshops*, 2010, pp. 1–8.

[5] M. Christen, O. Schenk, and H. Burkhart, "Patus: A code generation and autotuning framework for parallel iterative stencil computations on modern microarchitectures," in *IPDPS*, 2011, pp. 676–687.

[6] M. Perrone, L.-K. Liu, L. Lu, K. Magerlein, C. Kim, I. Fedulova, and A. Semenikhin, "Reducing data movement costs: Scalable seismic imaging on blue gene," in *IPDPS*, 2012, pp. 320–329.

[7] L. Lu and K. Magerlein, "Multi-level parallel computing of reverse time migration for seismic imaging on blue gene/q," in *PPOPP*, 2013, pp. 291–292.

[8] P. Micikevicius, "3D finite difference computation on GPUs using CUDA," in *GPGPU*, 2009, pp. 79–84.

[9] E. H. Phillips and M. Fatica, "Implementing the himeno benchmark with CUDA on GPU clusters," in *IPDPS*, 2010, pp. 1–10.

[10] K. Sano *et al.*, "Scalable streaming-array of simple soft-processors for stencil computations with constant memory-bandwidth," in *Proc. FCCM*, 2011.

[11] H. Fu and R. G. Clapp, "Eliminating the memory bottleneck: an FPGA-based solution for 3D reverse time migration," in *Proc. FPGA*, 2011.

[12] M. E. Wolf, D. E. Maydan, and D.-K. Chen, "Combining loop transformations considering caches and scheduling," in *MICRO*, 1996, pp. 274–286.

[13] J. Hull, *Options, Futures and Other Derivatives*, 6th ed. Prentice Hall, 2005.