A Configurable VHDL Template for Parallelization of 3D Stencil Codes on FPGAs

ERSA'12 Distinguished Paper

Franz Richter, Michael Schmidt and Dietmar Fey
Department of Computer Science, Chair of Computer Science 3 - Computer Architecture,
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

Abstract— *2D and 3D stencil code applications are very common in scientific computing, but their performance is mostly limited by the memory bandwidth. Elaborate on-chip buffering techniques minimize memory transfers, but they cannot be directly realized on fixed general-purpose processors or GPUs.*

FPGAs instead offer flexibility regarding the processing scheme, the degree of parallelism and the numerical representation of values. This enables FPGA-based problem-solvers to perfectly scale from low-power embedded devices to high-performance accelerators with a much higher performance-to-power ratio than conventional processing nodes. To reach optimal performance, elaborate buffering techniques within the FPGA are necessary to avoid redundant memory access, first of all in 3D space.

We created a generic VHDL template to ease development of 3D stencil-based applications, using Full Buffering to minimize data transfers. The template allows a custom number format, together with variable parallelization in space and time. The parameters can be set according to the capabilities of the underlying hardware and the requirements of the application.

Keywords: Stencil Codes, FPGA, Full Buffering

1. Introduction

In numerical approximations a specific problem is often discretized in both, space and time. It is therefore mapped on a regular grid and computing one step in time for a single grid point at a certain position, referred to as *cell*, can be put down to a function of itself and its surrounding [1]. To perform one iteration, this function, also called *stencil*, is applied to every cell of the grid, while several thousand iterations are often needed for convergence of the results.

Applications of these stencil codes include 2D image preprocessing operations, which we already have analyzed in detail [2]. In high performance computing (HPC), 3D stencils are often used for physical simulation, like the heat dissipation with the *Jacobi Iterator* [3] or a particle behavior with *Lattice Gas* or *Lattice Boltzmann* methods [4].

These applications are very data-intensive, first of all for the 3D problem space. Using off-the-shelf hardware like CPUs and GPUs, stencil codes are mainly memory bound, which means that the external memory bandwidth is the bottleneck in the processing chain.

Fortunately, this limit can be shifted by using clever strategies for buffering and parallelization on FPGAs, because they can be strongly customized to the problem, instead of having to adapt the algorithm to a fixed architecture. Furthermore, even high-performance FPGAs have a moderate power consumption, in contrast to normal CPUs and GPUs, which allows HPC applications to be realized efficiently [5].

1.1 Buffering

To circumvent the performance limit on FPGAs, extensive use of on-chip memory is necessary to avoid redundant memory access and allow parallelization of computation. The high amount of resources of today's FPGAs enables hardware designers to use *Full Buffering (FB)* in favor of *Partial Buffering (PB)* [6].

For PB, only the data needed by the current computation is stored to minimize memory consumption. Fig. 1a shows a three-dimensional problem space of size $M \times N \times O$. Each block represents one value and its shade gives its current role in computation. The light gray blocks have neither been loaded nor processed yet. Values which are needed by the current stencil update are medium gray. Here, an $X \times Y \times Z$-wide area is used, which is called neighborhood. To perform the next update, $Y \cdot Z$ dark gray blocks are needed, and as much as data is now obsolete and can be evicted (the most left column of the neighborhood).

Hence, for a single computation, several values have to be loaded from external memory, limiting the resulting performance.

In contrast to that, the main idea of FB is to store data internally until all computation relying on it has been performed. In Fig. 1b, these additionally stored values are shaded differently, compared to PB. Data has still to be loaded to perform an update, but it depends no longer on the size of the stencil. Instead, only a single value is needed, since all other data is already present.

During computation, the stencil is applied row-wise, from the top to bottom and plane-wise from the front to the back. Thus, a value must have been used for computation at every

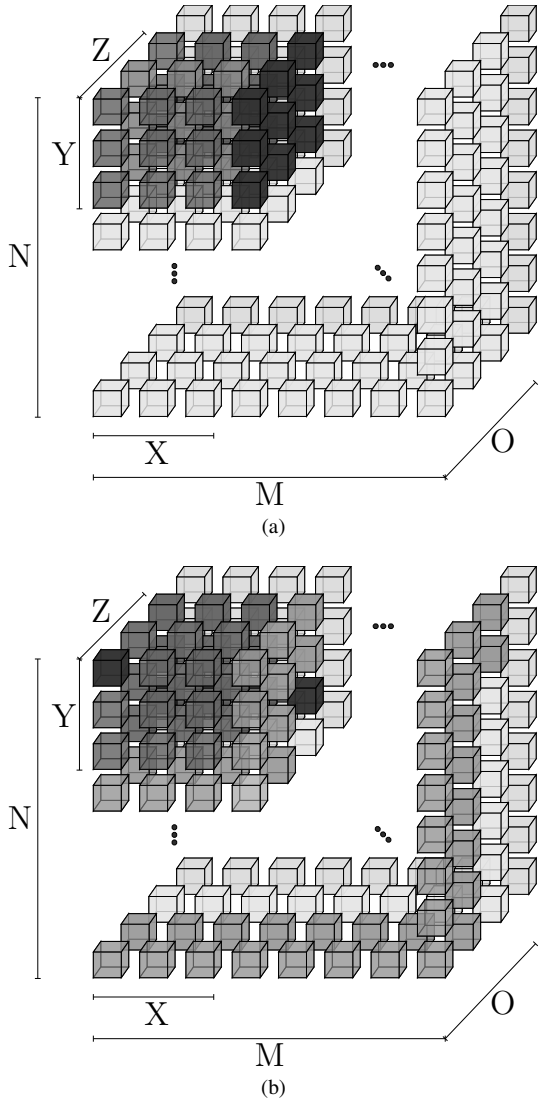


Fig. 1
3D PARTIAL BUFFERING (PB) (A) AND FULL BUFFERING (FB) (B)

position of the stencil, i.e $X \cdot Y \cdot Z$-times, until is can be evicted.

It is obvious that this is optimal regarding the utilization of external memory, at the expense of a high demand for local buffer space. However, this paper will show that FB is worth on modern FPGAs, since it is the only way to optimally minimize redundant memory access and it is furthermore predestined for stream processing, making it perfectly compatible with typical stencil codes.

The following sections give an overview of prior work concerning stencil codes on FPGAs and explain the motivation for our generic approach, before the template itself is explained.

1.2 Related Work

Stencil code applications on FPGAs are well discussed and several different architectures have been proposed.

Ref. [7] uses finite-difference schemes for real-time sound synthesis. They use a two-dimensional array of processing elements (PE), each containing an independent controller, internal memory (distributed or Block RAM) and a fixed-point arithmetical unit. Parallelism is achieved by using several PEs, which in turn can contain multiple and pipelined arithmetical units. All input data is spread over of the PEs, if enough on-chip space is available. Otherwise, external memory has to be used, limiting the total amount of PEs available since all PEs have to communicate simultaneously. For large inputs, it is not analyzed further how the memory bandwidth affects the overall performance.

The different degrees of parallelization are similar to ours, but at a different level of abstraction. We do not see any advantage of implementing multiple PEs, each with separate control logic and memory access, on a single FPGA. Instead, enlarging a single PE to the architectural limits significantly reduces the overhead for maintenance since a single controller is sufficient.

The authors of [8] [9] claim to have developed the first hardware accelerator card for the 3D finite-difference time-domain method. They build a memory hierarchy consisting of on-chip BlockRAM, coupled with on-board SRAM and DRAM, connected to PCI-bus of the host system. The BlockRAM is mainly used as input and output buffer to the external DRAM to allow bursts of data. Each arithmetical unit, of which more than one may exist for parallelization, accesses disjoint on-chip buffers and performs computation on single precision floating point numbers. They also independently exchange data with external memory, but we do not see multiple memory accesses in parallel with totally different addresses to be feasible.

In [10], an architecture for star-shaped stencil codes of size $n \times (n+1) \times n$, for even n, with single precision floating point data is proposed. It consists of the front-end, which is mainly responsible for external memory access and buffering of the data already loaded, and the back-end, implementing the actual update logic, running at twice the clock frequency of the front-end. The engines are controlled by the input FIFO and stalled if necessary. They rely on an extended PB, minimizing memory access along one axis. The paper emphasizes the speed of on-chip data transfers to maximize stencil throughput. This throughput depends on the clock frequency, the amount of BlockRAMs used for parallel data access and the degree of parallelization of computation, but we do not expect any of the three to be a bottleneck, despite of the external memory bandwidth, which is not primarily minimized in this approach.

The most recent publication [11] shows a similar approach to ours. They use an FB scheme for the 3D Reverse Time Migration algorithm, with the same argument as ours, that

the limitations by the external memory bandwidth is the main bottleneck in today's applications. They also cover different shapes of stencils, coming to the conclusion that compact ones are more suitable in FB-based applications due to less memory requirements, even if they have a higher computational intensity. Despite of our work, they do not give a detailed description of the hardware structure and the resource consumption for different configurations. Furthermore, they only consider 2D blocking to allow arbitrary problem spaces, though 1D blocking involves less redundancy, compared to 2D, as it is explained in Sec. 2.2.

1.3 Template

In contrast to the work listen above, our goal was to provide a flexible framework to ease the development of future 3D stencil-based streaming-applications on FPGAs, but offering the performance and transparency of a customized solution. This general approach is new to out knowledge. It is an advancement of our prior work on 2D stencil codes, for which we already have developed a VHDL template, firstly presented in [2].

The 3D template is based on the results of [12]. It can be adapted by different parameters, configuring the size of the problem space, the size of the stencil or the amount of bits per cell, to fit the application's needs. Furthermore, the degree of parallelization in both, space and time, can be adapted, according to the memory bandwidth and the FPGA-resources available. If the problem space is too large, partitioning may be necessary, imposing a certain degree of overhead. This scalability allows the template to be used in all ranges of complexity, from low-power embedded devices to HPC-accelerators, without having to bother with data handling.

The paper is structured as follows. The next section explains the components and parameters of our template. The third section presents some mathematical background, together with estimations and measurements of processing time and resource consumption. In the last section, we present a demonstrator design.

2. Template Design

The main goal of the template is to allow an optimal use of resources on an FPGA for a given problem, while taking full advantage of the external memory bandwidth. Stencil codes are usually very unbalanced regarding the amount of data required for a single update and the computational intensity. Therefore, they are often limited by memory bandwidth, first of all for 3D problems. Hence, an optimal buffering scheme is needed to optimize the overall performance.

2.1 Full Buffering for 3D Stencil Codes

FB reduces memory transfers to a minimum by keeping data inside of much faster on-chip memory as long as it is required for computation. On FPGAs, a limited amount of on-chip SRAM with a latency of a single clock cycle is available for buffering.

The total amount of values A_{FB}, which have to be buffered internally for an FB scheme, is given by (1) (see Fig. 1).

$$A_{\text{FB}} = M \cdot N \cdot (Z-1) + M \cdot (Y-1) + X \qquad (1)$$

Note that each value of the grid is represented by a constant amount of bits. In scientific computing, single- and double-precision floating point numbers are very common, though the wide range of values is often not needed by the application. In fact, a fixed-point number format may allow a much more hardware-efficient implementation of arithmetical units, and even raise the accuracy of values within the interval. Therefore, the parameter D is used for the width of a single value, instead of a fixed data type.

As mentioned above, the size of buffer space required A_{FB} depends on the stencil itself since each value has to be stored until is has been streamed through the stencil, starting at the bottom-right of the last dimension and ending at the top-left of the first dimension, if it is applied in a row- and plane-wise fashion. As a result, the space required is independent from the parameter O.

More complex problems or higher requirements on accuracy often lead to larger stencils, which results in a strong increase of buffer space. High order compact schemes[13][14] reduce the size of the stencil at the expense of computation, which is not regarded to be critical on FPGAs, despite of the capacity of on-chip memory.

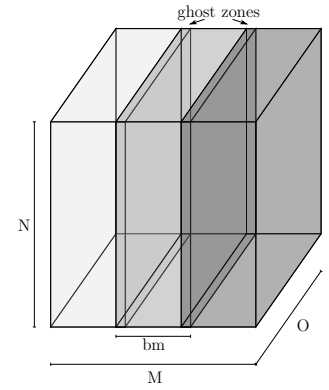
In a PB approach, solely the processed stencil of size $X \times Y \times Z$ is stored which requires much less data to be buffered by the FPGA, with the drawback that $X \cdot Y$ values have to be loaded to update a single value. This leads to a lot of redundant memory access in a PB scheme, which is therefore unfavorable if enough on-chip storage is available, as it is on current FPGAs.

Apart from PB and FB, we think it is not worth to implement a CPU-like memory hierarchy since caches are intended to speed up applications with an unpredictable memory access pattern, which is not necessary for streaming applications, since data is consumed sequentially.

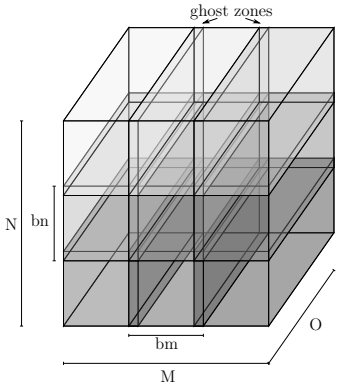
Elaborate techniques like multi-buffering, well known from cluster computing to hide memory latency, are also not able to speed up the FB-approach any further since the external memory bandwidth remains as bottleneck. In general, the actual data transfer between the template and external memory should make excessive use of sequential burst transfers, but since the exact conditions are user- and application-specific, data transfer is not to be handled by the template itself.

2.2 Blocking

Depending on the size of the stencil and the FPGA used, it is likely that an FB approach is not possible for



(a)



(b)

Fig. 2
SCHEME FOR 1D (A) AND 2D BLOCKING (B)

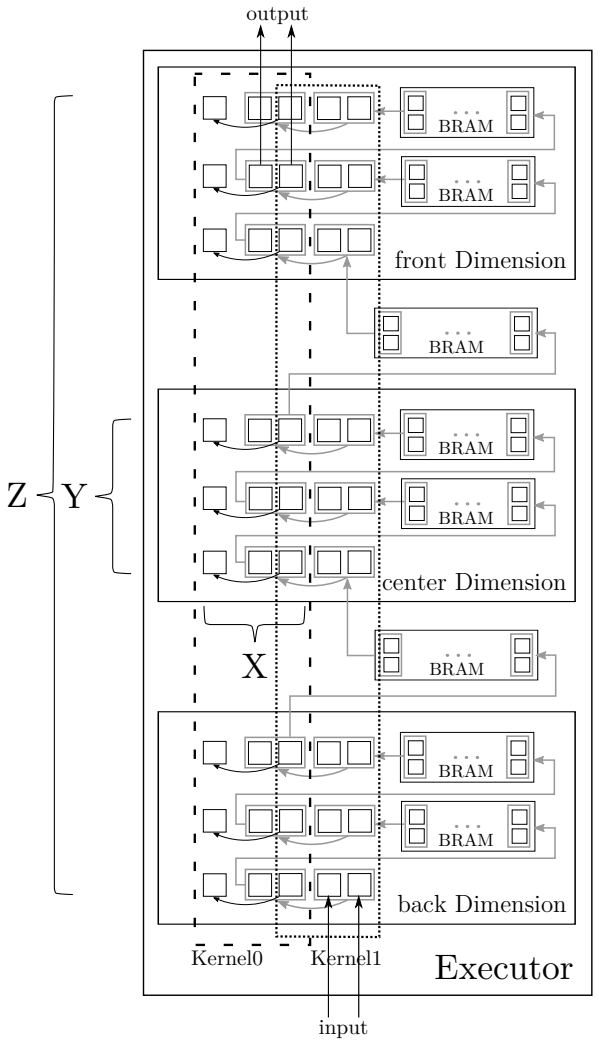


Fig. 3
SCHEMATIC OF FB TEMPLATE. EXECUTOR CONSISTS OF SEPARATE DIMENSIONS WITH INTERMEDIATE BUFFERS AND TWO KERNELS ($P = 2$) FOR A STENCIL OF SIZE $X = Y = Z = 3$

the complete input space. A sufficient compromise for the buffering scheme has to be found. The input space has to be split into blocks. FB is then applied to a block and all blocks are processed consecutively for each iteration. This is called *blocking*.

There are two main blocking methods. For 1D blocking as in Fig. 2a, the input space is split into slices, each spanning two whole dimensions. It is recommended for a non-square input space to choose O as the largest dimension to fully exploit the streaming character, because as shown in the section before, A_{FB} is independent of the parameter O. This maximizes the throughput and hides the initial wind-up overhead to fill the buffer. Using N for the smallest dimension reduces the size of the buffer and thus allows less blocks at all.

If the input space is even too large for 1D blocking to be feasible, which would result in many slices with a limited width bm, blocking in two dimensions is applicable (see Fig. 2b). Again, the largest dimension of the input space should be used for O.

On the other hand, blocking introduces redundancy because neighboring blocks must contain overlapping values in order to update the boundaries, called ghost zones.

This blocking mechanism can also be used to efficiently distribute the stencil computation of the input space to separate devices, if a multi-FPGA platform is used for example, which even does not have to be homogeneously due to the configurability of the template. For such a solution, the synchronization overhead is regarded to be negligible due to the deterministic character of the template. To minimize communication among separate FPGAs, it is applicable to enlarge the overlapping, even though it raises redundancy in memory transfers. That way, multiple iterations can be computed without having to communicate off-chip. These techniques are also well-known from the area of cluster computing using MPI and can be adapted to FPGAs.

2.3 Implementation

We implemented a generic VHDL design based on FB, allowing different degrees of parallelism, and supporting arbitrary problem sizes by spatial blocking. To be flexible, no vendor-specific library components are used but only behavioral descriptions, to make full usage of the synthesizer's inference capabilities. The main components of the template are *Kernel*, *Dimension* and *Executor*. Their dependence is illustrated in Fig. 3 and is explained in the following.

The Executor encapsulates the buffering and computation of the template. For each update, new data is streamed into the Executor, and results computed by the Kernel are streamed out. To be able to embed the template in different environments, it is controlled by a simple clock-enable signal which has to be activated if new data is available.

The Kernel implements the computation of the stencil and depends on the application. A Kernel has access to all data required for the realization of a stencil operation on a cell of the input space. It has to be implemented by the user and may contain either combinational or sequential logic, depending on the complexity of the algorithm. Updates of a cellular automata, e.g. for the realization of the *Lattice Gas* model, may be performed in a single clock cycle, while a lot of other stencil codes are more complex, leading to considerable critical paths. Adding intermediate registers allows pipelining of operations in order to increase the system frequency. Though this raises the latency of a single update operation, it does not limit the application's performance, because with each clock cycle, a new result is available, similar to pipelining of a microprocessor. The size of the kernel depends on the X, Y, and Z of the stencil and can be configured. Nevertheless, it is recommended to use minimal stencils if possible, even if they imply a higher computational intensity, to save buffer space.

Based on registers and FIFOs, an instance of the component Dimension holds the rows of an $M \times N$-wide plane which are currently covered by the Kernel. The remaining data of each plane is stored in a separate FIFO, except for the last plane, which directly consumes data from the Executor's input (see Fig. 1b).

In detail, each row within a Dimension contains X times D-wide registers and a separate $M - X$ values wide BlockRAM-FIFO. The registers are used as input to the Kernel, since FIFOs can not handle multiple reads simultaneously if they are realized with BlockRAM. On FPGAs, one cell of BlockRAM usually only has two ports. These registers are queued, allowing data to be shifted from one end to the other. The end of a queue is connected to the output port of a FIFO, which contains the remaining data of the row. On each update, the registers and FIFOs are shifted by one, allowing the next values to be processes by the Kernel. If a value is shifted out of the register queue, it is either sent to the row spatially above, the next plane, or, if it is not needed any more, it is discarded.

The explained buffering scheme allows to perform one update with a single load and store. This can be increased by introducing different degrees of parallelism, which is explained in the next section.

2.4 Parallelization

Depending on the amount of bits per value D and the external memory bandwidth, the degree of spatial parallelization P can be adapted. With single precision floating point numbers, 64 bit wide dual-port ram, and an the same clock frequency for FPGA and memory, $P = 2$ allows to exploit the available bandwidth. Two ports are needed to read and write data simultaneously.

By setting P, several neighboring values are updated simultaneously with only $P - 1$ additional values that have to be buffered. The demand for processing logic increases due to multiple Kernels, but this is not a limiting factor for common stencils, so P is only limited by the external memory bandwidth.

Especially for applications with small D, it leads to a significant rise of performance. This mechanism also benefits from the fact that the data format used on the FPGA is fully customizable.

If P is set properly, the only way to improve performance is to reduce the amount of data to be transferred by temporal parallelization of computation, called pipelining. Multiple iterations are computed per sweep by queuing up I Executors. Each Executor has its own FB of the current region and enhances the iteration by one. It's output is used as input to the next one, again enhancing the iteration. In theory, it is possible to compute all iterations with only a single load and store per value, but multiple Executors raise the demand for on-chip memory by factors. Pipelining also raises the latency of the template, but it is tolerable if a high amount of data is processed.

If blocking is required due to the size of the problem space, the demand for memory even grows. For every iteration to be processed within one sweep, additional values surrounding the current block are required, called . Because the block size of $bm \times bn \times O$ has to be set with regard to the available internal memory, the actual data block is smaller if more than one iteration is realized.

Several ways to configure the template have been introduced by now. All of them, the variable blocking mechanism, i.e. 1D or 2D, the block size, the amount of P neighbors computed in parallel or the depth of the Executor-pipeline I either depend on the on-chip resources available or the external memory bandwidth or both. It is an optimization problem to find optimal parameters for an actual problem. For 2D stencil code applications, we have already analyzed such optimization in more detail [15] and developed an analytical model. At the moment, we are working on an advanced version of our model to get an optimal parameter set also for 3D.

Table 1
PARAMETER DEFINITION

Identifier	Description
$M \times N \times O$	Width, height, depth of input with border
$m \times n \times O$	Size of block with data to be processed
$bm \times bn \times O$	Size of block, with ghost zones
k	Number of blocks
$X \times Y \times Z$	Size of the stencil
D	Bits per value (per cell)
P	Spatial parallelization
I	Temporal parallelization
A_{FB}	Buffered cells for Full Buffering
A_{I}	Buffered cells with pipelined Executors
c_{block}	Total memory accesses with blocking
c_{FB}	Total memory accesses without blocking
PF_{BL}	Performance of blocking
PF_{PB}	Performance of Partial Buffering

As a rule of thumb, P is set as mentioned above. The problem space is split into equal blocks that barely fit into the FPGA. If resources are still available, I is increased.

3. Evaluation

In this section, the mathematical background of the template is analyzed. Furthermore, syntheses results and the performance are presented. Tab. 1 gives an overview of the parameters of the template. Most of them have been introduced by now.

3.1 Overhead of Blocking

Based on 1, we determine the performance of blocking in relationship to an FB of the complete input space and compute the amount of redundant memory access.

The size of the ghost zone depends on the stencil size. For simple $X = Y = Z = 33$ stencils the width of the ghost zone is one for a single iteration. The block size $bm \times bn$, which also has to contain the required ghost cells, has to be determined by D and the available on-chip memory, according to (1). The size of the block $m \times n$ of actual processed values can then be determined as shown in (2).

$$\begin{aligned} m &= bm - X + 1 \\ n &= bn - Y + 1 \end{aligned} \tag{2}$$

Depending on the size of the input space, the number of blocks k, which have to be processed consecutively, can be determined by (3).

$$k = \frac{M \cdot N}{m \cdot n} \tag{3}$$

The total amount of memory accesses for FB on the whole input space, c_{FB}, can be determined by (4), and with blocking, denoted as c_{block}, by (5). Note that for 1D blocking, bn becomes N, reducing redundancy.

$$c_{\text{FB}} = k \cdot m \cdot n \cdot O \tag{4}$$

$$c_{\text{block}} = k \cdot bm \cdot bn \cdot O \tag{5}$$

For FB, all redundant memory accesses are eliminated, leading to an optimal performance. The ratio between c_{FB} and c_{block}, giving the fraction of performance that remains due to redundancy of the ghost areas, is ascertained by (6).

$$PF_{\text{BL}} = \frac{c_{\text{FB}}}{c_{\text{block}}} = \frac{m \cdot n}{bm \cdot bn} = \frac{(bm - X + 1) \cdot (bn - Y + 1)}{bm \cdot bn} \tag{6}$$

Eq. (6) proves our expectations. A greater internal memory allows a greater block size $bm \times bn$ and, hence, a higher performance because the amount of data needed for the ghost area is strongly dominated by the data enclosed. Furthermore, a greater stencil size implies a greater ghost zone width, leading to an increase in redundancy and, thus, to a lower performance.

Another interesting issue at this point is the performance of blocking with regard to PB, where (7) values have to be loaded.

$$PF_{\text{PB}} = \frac{M \cdot N \cdot O}{M \cdot N \cdot O \cdot X \cdot Y} = \frac{1}{X \cdot Y} \tag{7}$$

From the equations it follows that if FB is not possible, blocked FB is still to be favored over PB since $PF_{\text{PB}} < PF_{\text{BL}}$ for all $bm > X$ and $bn > Y$.

3.2 Pipelining

It was already mentioned above that for pipelined Executors, the actual block size $m \times n \times O$ has to be wider to be able to update border values correctly. For I iterations, a block has to contain $(m + I \cdot (X-1)) \times (n + I \cdot (Y-1)) \times O$ values. For high I, it is feasible to use the template with different parameters for every pipeline stage and omit the outer border of the previous stage while streaming.

The amount of buffer entries of size D for a Executor pipeline of depth I is then given by (8).

$$\begin{aligned} A_{\text{I}} = \sum_{i=0}^{\text{I}-1} \Big(&(m + i(X-1)) \cdot (n + i(Y-1)) \cdot i(Z-1) \\ &+ (m + i(X-1)) \cdot i(Y-1) + i(X-1) + 1 \Big) \end{aligned} \tag{8}$$

This shows why increasing the pipeline depth highly increases the buffer space.

3.3 Synthesis

By now, only theoretical assumptions were made. To get real numbers, the template was synthesized with Xilinx ISE 11.5 for a Xilinx Virtex-5 XC5VLX110T FPGA. The FPGA contains 69120 Flip-Flops (FFs) and Lookup-Tables (LUTs) respectively, and 148 BlockRAM cells with a capacity of 2KiB each.

Table 2
SYNTHESIS RESULTS

$M \times N \times O \cdot P \cdot I$	FFs	LUTs	BlockRAM	MHz
$32 \times 32 \times 32 \cdot 1 \cdot 16$	11328	2113	80	317
$32 \times 32 \times 32 \cdot 1 \cdot 28$	19824	3697	140	317
$64 \times 64 \times 64 \cdot 1 \cdot 1$	728	157	11	303
$64 \times 64 \times 64 \cdot 2 \cdot 4$	4816	496	80	308
$64 \times 64 \times 64 \cdot 4 \cdot 4$	7088	416	128	316
$128 \times 128 \times 128 \cdot 1 \cdot 1$	748	209	35	301
$128 \times 128 \times 128 \cdot 4 \cdot 1$	1782	1860	53	282
$128 \times 128 \times 128 \cdot 1 \cdot 4$	2992	833	140	301
$256 \times 256 \times 256 \cdot 1 \cdot 1$	770	239	131	298
$256 \times 256 \times 256 \cdot 2 \cdot 1$	1228	208	140	306
$256 \times 256 \times 256 \cdot 4 \cdot 1$	1814	168	152	257

These BlockRAMs are predestined to be used for buffering data on-chip, but they are less flexible than Lookup-Tables. Large values for D and P could lead to a higher demand on blocks needed due to the limited interface-width of 36 bit of each port. If the target architecture is fixed, technology-dependent optimization of allocated resources may reduce the demand. For fixed parameters of the problem space and the stencil, even further optimization is possible, e.g. by implementing the shorter row-buffers as distributed RAM.

Tab. 2 shows the results for different parameters with $D = 32$. It can be seen that the main limitation for the template is the available on-chip memory, as already explained before. Even a small input space of $256 \times 256 \times 256$ almost exhausts the capacity, leaving no room to increase I but only P. Hence, for large input spaces, blocking is inevitable to achieve a notable speedup.

3.4 Performance

Tab. 3 shows some performance estimations. The runtime is based on a clock frequency of 400 MHz for a block size of $M = N = O = 128$, and a stencil of $X = Y = Z = 3$. Note that the actual frequency depends on the speed of the FPGA and the complexity of the Kernel. To get a reasonable runtime, 1000 iterations are assumed. The first column shows the total amount of memory transfers for reading and writing, followed by the on-chip space needed in multiples of D. The runtime is estimated for a dual-channel $64bit$ wide memory interface with $D = 32$.

It can be seen that an optimal buffering strategy like FB can greatly reduce the runtime for 3D stencil code applications. Furthermore, the speedup can be highly increased by spatial and temporal parallelization. Compared to regular FB, the asymptotic speedup with optimization is $P \cdot I$.

It is necessary to clarify that the accuracy of performance estimations of hardware descriptions differ from theoretical peak performance of CPUs and GPUs because it is totally deterministic and thus predictable what happens during one clock cycle. Therefore, the performance of the template can be exactly given without benchmarking. Of course, for an actual application, additional components are needed, but the memory interface remains the bottleneck.

Table 3
COMPARISON OF DIFFERENT CONFIGURATIONS

	r|w $\cdot 10^9$	Buffer	Speedup	Runtime $[s]$
No buffering	54.0| 2.0	27	1	140
PB	18.0| 2.0	27	2.8	50
FB	2.1| 2.0	33027	13.7	10
$P = 2$	1.1| 1.0	33028	26.7	5
$P = 2$, $I = 2$	0.6| 0.5	66056	50.9	3
$P = 2$, $I = 4$	0.3| 0.25	132112	180.0	1.4

Taking power consumption into account, the FPGA-based stencil solution shows its actual potential. The FPGA in our design needs about 1.1 W, independent of the actual problem.

An optimized implementation of the Jacobi-iteration for Laplace's equation on recent Nvidia Fermi and Geforce GTX GPGPUs delivers a performance of 10 GLUPS[1] per 250 W or 40 MLUPS/W [16] for double-precision floating point operations.

Based on the maximum clock frequency of approximately 100 MHz, 2.8 GLUPS for double precision can be reached on a small input space of size $32 \times 32 \times 32$ (see Tab 2, leading to an efficiency of 2.5 GLUPS/W, which is magnitudes higher than the GPGPUs, under the assumption that for each clock cycle, new data is available, which can easily be fulfilled due to the low frequency.

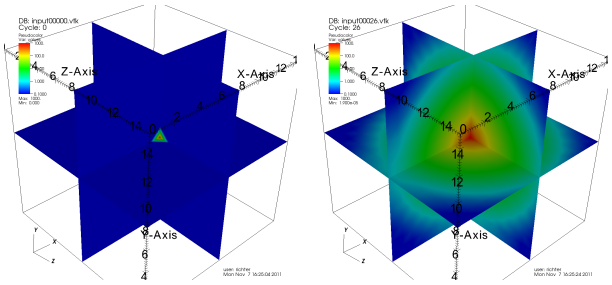
For larger problems, the performance drops, but even 0.2 GLUPS/W for $P = 2$ on a grid of size $256 \times 256 \times 256$ outperforms the GPUs by far. By using multiple FPGAs, power consumption scales linear with performance, allowing to build „green" HPC-systems. A very interesting example for the potential of FPGAs for high performance is [17], where a similar approach is used, but totally optimized for high-performance streaming applications and at a much higher degree of abstraction.

Furthermore, the FPGA used here is of medium size and two generations behind current versions. For greater applications, a larger FPGA offers even higher performance due to more on-chip memory, with only a limited increase of power consumption.

This analysis illustrates that FPGAs are able to outperform GPGPUs by factors with regards to power efficiency.

The comparison of the power consumption of the single FPGA-chip and the whole GPGPU-board is seen to be reasonable since all components used for computation are taken into consideration, e.g. the on-board RAM of the GPGPUs vs. the FPGA's on-chip BlockRAM.

[1] Giga Lattice Updates Per Second, conforms to single stencil applications per second



(a) $i = 10$ (b) $i = 26$

Fig. 4
3D JACOBI-ITERATION FOR CENTRAL HOT SPOT

4. Reference Design

We implemented a demonstration application based on the Jacobi-iteration to show the applicability and flexibility of our template-based approach.

FloPoCo [18], an open and free generator for arithmetic cores, was used to create the Kernel. The resulting FPU has a pipeline depth of 13 cycles, and allows a target clock rate of 100 MHz on the FPGA mentioned previously. If the FPU is split up into more stages, a higher frequency is possible. With only about 5 % of available resources of the FPGA, the Kernel allows a high degree of parallelism.

The results of the demonstrator can be seen in Fig. 4. With no loss of generality, an input space of only $16 \times 16 \times 16$ was used. *Visit*[2] was used to visualize the 3D volume.

5. Conclusion and Outlook

We presented a generic VHDL template for 3D stencil-based applications, exploiting Full Buffering to minimize external memory accesses and increase performance of memory bound applications.

To configure the template, several parameters can be set, according to the problem and target architecture. Execution can be accelerated by spatial and temporal parallelization, depending on the memory bandwidth and the amount of on-chip storage available.

The main advantage of the template is its flexibility. Due to minimized demands to the environment, it is possible to embed it in a large variety of settings. This may be, for example, a low-power embedded streaming application, or a PCIe-based multi-FPGA accelerator for high performance computations.

Together with the mathematical model we are developing, we will be able to find an optimal mapping of a given stencil-based streaming application to the resources of a system and to implement it without having to bother with data access, while preserving the advantage of efficiency of FPGA designs.

[2] https://wci.llnl.gov/codes/visit/

References

[1] M. Sadiku, *Numerical techniques in electromagnetics*. CRC Press, 2000.

[2] M. Schmidt, M. Reichenbach, A. Loos, and D. Fey, "A smart camera processing pipeline for image applications utilizing marching pixels," *Signal & Image Processing: An International Journal (SIPIJ)*, vol. Vol. 2, no. No. 3, pp. 137–156, 9 2011.

[3] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, ser. Computational Science. Taylor & Francis, 2010.

[4] D. Wolf-Gladrow, *Lattice-gas cellular automata and lattice Boltzmann models*. Springer, 2000.

[5] M. Herbordt, T. VanCourt, Y. Gu, B. Sukhwani, A. Conti, J. Model, and D. DiSabello, "Achieving high performance with FPGA-based computing," *Computer*, vol. 40, no. 3, pp. 50 –57, march 2007.

[6] X. Liang, J. Jean, and K. Tomko, "Data buffering and allocation in mapping generalized template matching on reconfigurable systems," *J. Supercomput.*, vol. 19, pp. 77–91, May 2001.

[7] E. Motuk, R. Woods, S. Bilbao, and J. McAllister, "Design methodology for real-time fpga-based sound synthesis," vol. 55, no. 12, pp. 5833–5845, 2007.

[8] J. P. Durbano, F. E. Ortiz, J. R. Humphrey, M. S. Mirotznik, and D. W. Prather, "Hardware implementation of a three-dimensional finite-difference time-domain algorithm," vol. 2, pp. 54–57, 2003.

[9] J. P. Durbano, F. E. Ortiz, J. R. Humphrey, P. F. Curt, and D. W. Prather, "FPGA-based acceleration of the 3D finite-difference time-domain method," in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 156–163.

[10] M. Shafiq, M. Pericas, R. de la Cruz, M. Araya-Polo, N. Navarro, and E. Ayguade, "Exploiting memory customization in FPGA for 3D stencil computations," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, dec. 2009, pp. 38 –45.

[11] H. Fu and R. G. Clapp, "Eliminating the memory bottleneck: an FPGA-based solution for 3d reverse time migration," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, ser. FPGA '11. New York, NY, USA: ACM, 2011, pp. 65–74.

[12] F. Richter, "A full buffering based template for 3d stencil code applications on FPGAs," Diploma Thesis, Department of Computer Science, Chair of Computer Architecture, Friedrich-Alexander-University Erlangen-Nuremberg, 2011.

[13] W. F. Spotz, "High-order compact finite difference schemes for computational mechanics," Ph.D. dissertation, University of Texas at Austin, dec, see http://www.cisl.ucar.edu/css/staff/spotz/papers/Dissertation/Spotz_Dissn.pdf.gz, visited 2011-09-22.

[14] G. Sutmann and B. Steffen, "High-order compact solvers for the three-dimensional poisson equation," *Journal of Computational and Applied Mathematics*, vol. 187, no. 2, pp. 142 – 170, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377042705001500

[15] M. Reichenbach, M. Schmidt, and D. Fey, "Analytical model for the optimization of self-organizing image processing systems utilizing cellular automata," in *SORT 2011: 2nd IEEE Workshop on Self-Organizing Real-Time Systems*, Newport Beach, 2011, pp. 162–171.

[16] A. Schäfer and D. Fey, "High performance stencil code algorithms for GPGPUs," *Procedia Computer Science*, vol. 4, no. 0, pp. 2027 – 2036, 2011, proceedings of the International Conference on Computational Science, ICCS 2011.

[17] O. Lindtjorn, R. Clapp, O. Pell, H. Fu, M. Flynn, and O. Mencer, "Beyond traditional microprocessors for geoscience high-performance computing applications," *IEEE Micro*, vol. 31, pp. 41–49, March 2011. [Online]. Available: http://dx.doi.org/10.1109/MM.2011.17

[18] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with flopoco," *Design Test of Computers, IEEE*, vol. 28, no. 4, pp. 18 –27, july-aug. 2011.