Information and Communication Technologies (ICT) Programme

Project N°: FP7-ICT-287804

D1.3: Definition of Evaluation Criteria

Author(s): Oliver Pell (MAX), Mathieu Robart (STM), Andreas Brokalakis (SYN), Tobias Becker (IMP), Karel Bruneel (GNT)

Status -Version: Version 1.0 (Final)

Date: August 27, 2012

Distribution - Confidentiality: Public

Code: FASTER_D1_3_MAX_FF-20120827

Abstract:

This document outlines the evaluation procedure and criteria that will be adopted in the FASTER project.

Disclaimer

The FASTER Consortium is the following:

Beneficiary Number	**Beneficiary name**	**Beneficiary short name**	**Country**
1(coordinator)	Foundation for Research and Technology – Hellas	FORTH	Greece
2	Chalmers University of Technology	CHT	Sweden
3	Imperial College London	Imperial	UK
4	Politecnico di Milano	PoliMi	Italy
5	Ghent University	UGent	Belgium
6	Maxeler	Max	U.K.
7	ST	ST	Italy
8	Synelixis	Syn	Greece

Document Revision History

Date	Issue	Author/Editor/Contributor	Summary of main changes
June 18, 2012	0.1	Oliver Pell	Initial draft, ToC
July 25, 2012	0.2	Mathieu Robart	STM contribution
August 1, 2012	0.3	Oliver Pell, Andreas Brokalakis	General criteria and SYN contribution
August 1, 2012	0.4	Oliver Pell	Adding baselines and general performance section
August 3, 2012	0.5	Andreas Brokalakis, Mathieu Robart, Oliver Pell	Notes on Section 2, proper modifications in NIDS section. Added ray tracing baseline.
August 7, 2012	0.6	Oliver Pell	Incorporating comments from Tobias Becker and Georgi Gaydadjiev
August 27, 2012	1.0 (Final)	Oliver Pell, Mathieu Robart, Andreas Brokalakis	Minor revision to all sections after Quality Control

Table of contents

1. Introduction 5
2. General Evaluation Criteria 5
2.1. Dynamic Configuration Cost 5
2.2. Area/size impact 6
2.3. Clock frequency impact 6
2.4. Performance 6
2.5. Functional density 6
2.6. Energy and power consumption 7
2.7. Design effort 7
2.8. Compile resources for primary bitstream and partial bitstreams 7
3. Domain Specific Evaluation Criteria 8
3.1. Embedded domain test case: Network Security 8
3.2. Desktop domain test case: Raytracing Application 10
3.3. HPC domain test case: Reverse Time Migration 12
4. Conclusions 14
5. References 14

1. Introduction

The FASTER project aims at introducing a complete methodology that allows designers to easily *implement and verify* a system specification on a platform that includes one or more general purpose processor(s) combined with multiple acceleration modules implemented on FPGA (one or multiple), taking as input a high-level description and fully exploiting, both at *design time* and at *run-time*, the capabilities of (partial) dynamic reconfiguration.

The main project's objective is to enhance the design of computing systems by including reconfigurability as an explicit design concept, with appropriate tools to support this.

This document describes the evaluation criteria that will be used to assess the FASTER toolchain both in the general sense and for the three industrial application case studies that are targeted. These applications cover different targeted application fields, from high-performance computing (HPC), workstation-class processing such as rendering, to embedded computations.

2. General Evaluation Criteria

The main advantage of utilizing partial reconfiguration is to reduce the FPGA resources required to implement a set of operations (since otherwise the operations might have to exist in parallel on the chip). In the extreme case, without partial reconfiguration the available chip sizes might not be large enough to support instantiation of all the required operations, so partial reconfiguration is essential to enable the application to be practically realized.

Another potential advantage is improved clock frequency, as compute kernels can be specialized to a certain condition, removing redundant logic and reducing propagation delay. A change in the required functionality or the specialization parameter will trigger reconfiguration and the time overhead of reconfiguration needs to be considered, since the reconfiguration operation requires some time to execute.

The following subsections highlight the way in which the expected benefits, restrictions and costs should be evaluated.

2.1. Dynamic Configuration Cost

Ideally a partially reconfigurable system should be able to function in the same way as a system that has all the functional units instantiated in hardware. Practically, this means that the time spent reconfiguring parts of the chip should be minimized, while the reconfiguration management system should be able to "mask" as much of this time as possible by proper scheduling. These can be quantified by the following:

1. Time required to reconfigure a certain amount of the chip that cannot be hidden or masked by other operations and results in system down time.
2. Percentage of the remainder of the chip that can continue running during reconfiguration.

The reconfiguration time is usually proportional to the area or amount of logic under reconfiguration as well as the throughput of the configuration interface.

We will measure (a) time spent performing configuration, (b) reconfiguration time that was not masked by other operations and (c) what percentage of the end-to-end time the time *(b)* represents.

2.2. Area/size impact

Reconfiguration should reduce the overall area for a number of processing operations, however it may have a negative area impact on each individual functional unit - for example, the area may need to be 'rounded up' to the size of a reconfiguration region. Area overhead can also be caused by additional infrastructure that is needed for the reconfigurable system. This area overhead should be minimized and can be assessed by comparing the area of operations on chips with and without reconfiguration support.

Area should be measured both in terms of FPGA resources (LUTs, FFs, DSPs, BRAMs) and also in terms of the smallest bounding box that can contain the operation logic.

Size reduction of the overall design may not always a relevant goal. If, for example, the available target device is fixed then there will be no benefit in downsizing the design. Instead, area reductions can then be used to increase the overall design efficiency or performance by fitting more operational units in the targeted device.

2.3. Clock frequency impact

As with area, it is possible that the use of reconfiguration will have an impact on the maximum clock frequency that an operation can run at. This impact will most likely be positive (since the operation no longer needs to coexist with other operations that would compete for e.g. routing resources). It is also possible to specialize reconfigurable components to a set of conditions which can allow removing some redundant logic, shortening the critical path. The clock frequency impact could also be negative due to quantization effects reducing the level of optimizations that can be applied.

This impact should be assessed for each reconfigurable operation, by comparing the clock frequency of (a) the operation in isolation and (b) the operation on a 'full chip' for both the reconfigurable and static case. Clock frequency will be measured as an absolute value and percentage change due to FASTER compared to the baselines.

2.4. Performance

The purpose of utilizing the FASTER flow will be to improve the performance characteristics of an application, in the broadest sense. The specific performance metric that should be measured specific to a particular application: for example, in some cases it might be overall throughput (e.g. operations per second), or latency to complete a certain operation. Performance is a related but not identical characteristic to clock frequency (2.3), and it is typically a function of clock frequency combined with other factors such as level of parallelism.

In all cases, some measure of overall application performance should be evaluated.

2.5. Functional density

As well as overall performance, another metric to consider is *efficiency*. The functional density of a system is the number of operations carried out divided by the number of resources allocated to performing those operations:

$$D = \frac{N}{AT}$$

where N is the number of useful operations (defined as *performance* in section 2.4 above), A is the area devoted to the problem, and T is the amount of time that area was devoted for.

We will evaluate functional density for the FASTER test cases, using a suitable application-specific operation-count or performance value for *N*, whole chip area for *A* and wall-clock time for *T*.

2.6. Energy and power consumption

The power consumption impact of the FASTER flow will be evaluated, as well as the overall energy impact. It is possible that the use of partial reconfiguration could increase or decrease power and energy: while the reconfiguration process itself incurs a power and energy overhead, the reconfigurable components are most likely more power and energy efficient.

For certain types of operation (e.g. fixed calculations), it is possible that the use of partial reconfiguration could increase the power consumption of a system (for example since a more of the chip could be active at a particular point in time since unused sections have been reconfigured into useful functions). Overall energy use could actually drop in this scenario as runtime is reduced by more than the increase in power consumption (since *energy = power × time*).

Alternatively in other use cases, the FASTER flow may reduce the power consumption of a system by replacing power-consuming logic to select between different possibilities (e.g. multiplexers) with reconfiguration between those possibilities.

In some applications that require some form of reconfiguration, energy could also be saved by only reconfiguring small parts of a chip rather than the full chip area.

The design can be evaluated in terms of average power consumption or energy per computation.

2.7. Design effort

Design effort is hard to measure, however it is also the case that high design effort is one of the main reasons for the limited exploitation of dynamic partial reconfiguration in commercial FPGA designs today.

We can approximately measure programmer effort as the number of programmer weeks to implement a particular application. Depending on the implementation flow followed this could involve measuring:

1. Time to implement a non-reconfigurable version of an application and the time spent to implement a separate reconfigurable version of the application
2. Time taken to extend an existing FPGA application to support partial reconfiguration
3. Time spent to implement the FPGA version of an application, compared to time taken to implement the original CPU code.

Towards the end of the project we aim for the academic partners to task parallel student groups to design the same function with and without the FASTER toolchain to help assess the design effort benefit.

2.8. Compile resources for primary bitstream and partial bitstreams

One of the major drawbacks of FPGA technology is the high compilation time, most of which is spent executing the map, place and route stages of the compilation flow. We will evaluate the impact of the FASTER toolchain on the compilation resources for FPGA bitstreams, both for "full" bitstreams and "partial" bitstreams.

It is possible that the FASTER toolchain may introduce additional parallelization options to the FPGA compilation process allowing the overall compilation resource use to increase while the end-to-end wall clock time is decreased.

Thus, compilation resources will be measured as both wall clock time on a specified test machine, and overall resource use in terms of CPU core or memory hours i.e. the integral over compilation runtime of the number of CPU cores or amount of memory utilized.

3. Domain Specific Evaluation Criteria

3.1. Embedded domain test case: Network Security

The Network Intrusion Detection application (NIDS) is a network security application and as such system designers and administrators need to respond to changes in a fast and efficient way. According to document D1.2 [1], the changes that have to be accommodated and require reconfiguration are the following:

- Small incremental updates may be required to add, change or expand certain IP addresses or address ranges that appear in existing detection rules
- New static pattern rules may have to be added to the static rule set or changes to the current patterns included in the rule set may have to be applied
- Updates in the existing regular expressions to cover more cases or correct mishandling of certain detection rules
- New regular expressions may have to be added
- Overall updates to the system may have to be performed in case of new policies or large-scale update to the operation of the NIDS system.

3.1.1. Baseline for Comparison

FPGAs are attractive to implement security related applications since they can offer higher performance compared to software solutions on one hand, and on the other compared to hardwired solutions they can be reprogrammed on the field, and therefore can adapt to either necessary updates or post-deployment discovered vulnerabilities. The approach that has been followed for the NIDS system is that of either full or static partial reconfiguration whenever an update or change is required as described above.

As a result, we will evaluate the FASTER proposed approach against the baseline of an FPGA implementation utilizing full reconfiguration whenever an update is required.

3.1.2. Update Design Effort and System Down - Time

The list presented in the beginning of this section is based on the frequency of each change that is expected to happen during system lifetime (i.e. small incremental updates are expected to be frequent while overall policy updates are relatively rare). This observation defines a number of evaluation criteria for both the design process and the actual reconfigurable system.

SYN identifies that the frequent small incremental updates can be serviced through micro-reconfiguration methods proposed in FASTER. From a design perspective, the time required to complete the changes should be faster compared to a complete module redesign as a result of parameterized hardware generation tools. Furthermore the generation of new configuration information has to be fast – much faster than the traditional generation of partial and full bitstreams. On the other hand, since these updates are frequent, the verification process also has to

be fast and more importantly it has to be accurate in the sense that incremental changes do not break the established functionality of the system (that is quickly verifying that the new updated system is at least able to detect all threats it could detect prior to the update).

From the system perspective, since the micro-reconfiguration process is expected to be frequent, the actual reconfiguration time has to disrupt system operation for a minimum amount of time. It should be noted that there is no point in making the process dynamic (that is keeping the parts of the system that are not updated, functioning) so as to cover part of the delay. That is because keeping certain hardware blocks offline (in order to update them) would result in compromising the effectiveness of the security system as a whole, since it would not be possible to detect the threats that their discovery rules are implemented in those specific hardware blocks. Therefore each update is associated with system down-time and this has to be kept minimal. This becomes even more crucial in the case of region-based reconfiguration that is associated with more profound changes in the system functionality or the addition of new features (such as new static rules or new regular expressions).

From the above, in order to evaluate the effectiveness of the FASTER approach two metrics can be taken into consideration:

1. Design time to incorporate changes or updates and new bitstream generation time (can be measured as CPU time on the development machine)
2. System down-time (can be measured as the overall wall-clock time that is required to apply a number of changes/updates to the NIDS system or the effective bandwidth – packets per second – over a period of time that covers different types of updates)

3.1.3. Overall Area Overhead

As mentioned in Section 2.2, the partial reconfiguration approach has a certain area impact. In the case of the NIDS system, the area overhead has to be minimal, since the system is an expansive one (functional blocks are generally not replaced only potentially modified, while new functional blocks have to be added periodically in order to accommodate new static rules or regular expressions). Thus area quickly becomes a precious resource.

In this case, the system is to be evaluated against a full system redesign that is not restricted by partial reconfiguration restrictions.

3.1.4. Performance

Since the NIDS system is extremely reliant on its performance (the NIDS depends on its performance to work properly and protect itself from certain types of attacks [2]), the impact of using a reconfiguration approach should be minimal. The original (non-partially reconfigurable) system is designed so that it can sustain a certain level of throughput (measured in packets per second) related to the network links that have to served. The proposed reconfiguration approach should have minimal overhead in the system performance (related to maximum frequency achieved by the design), so that larger designs wont be required in order to achieve the same level of performance (by augmenting parallelism for example). Note that according to Section 3.1.3, area is a major restriction and therefore it is not acceptable to resolve to solutions that require more area resources in order to compensate for lower clock frequencies.

The two basic metrics that can be used in this case are:

1. Clock frequency
2. System throughput

3.2. Desktop domain test case: Raytracing Application

The raytracing application is a very versatile solution that permits to simulate the interactions of light into a 3D synthetic scene and to render a view of this scene on screen, with potentially photorealistic results.

The core of the raytracing algorithm is well defined for several decades, but each application differs with the methods employed to compute intersections, propagate rays, represent 3D surfaces, accelerate traversals, or even evaluate reflectance and transmittance. For that reason, there is no reference benchmark that can be used to compare or evaluate the results produced by a single application.

3.2.1. Baseline for Comparison

In order to evaluate the benefits of the FASTER toolchain, the original C application can be used as reference benchmark, in term of image quality, memory footprint and computational performances.

The C code doesn't require any specific library or hardware acceleration, and should be easily ported on any kind of platforms required for this comparison phase (FPGA for example). We will run the C application on a suitable multi-core desktop CPU system.

3.2.2. Image quality

The first evaluation criterion is a subjective one, based on visual comparison and judgment of the final rendering. Only a human eye can define what is photorealistic and what is not. However, beyond this non-deterministic evaluation, reference images obtained from a trusted renderer can be used. In the FASTER case, the original application can be used to generate these reference images, and comparisons can be made at pixel level in order to evaluate the accuracy and fidelity of the optimized code once processed by the FASTER toolchain. This can be done using binned histogram comparison for example for similarity evaluation, or sum of absolute differences at pixel level.

Indeed, the quality of the resulting image can differ due to differences in floating-point precision in the computation of intersections or shading for example, or due to the stochastic approach used in sample selection, efficiently exchanging aliasing for noise in pixel and shadow evaluation.

3.2.3. Rendering speed

The second evaluation criterion is speed. As raytracing involves a high number of arithmetic operations for each pixel rendered (characteristic from a Desktop-class application), real-time is not yet achievable but for only simple scenes. However, rendering times can be compared against the reference application, for various scene complexities. As a consequence, difference scenes can be defined in order to stress one particular aspect of the application, and used as relative acceleration benchmarks:

- Basic: a simple scene with few primitives, a single light source, no reflection
- Geometry: a scene with a high number of primitives, simple materials, a single light source
- Illumination: a scene with few primitives, multiple light sources of difference areas
- Material: a scene with few primitives, a single light source, a complex material model
- Complex: a scene with a high number of primitives, complex materials, multiple light sources, high depth complexity
- Etc.

The rendering time and the resulting image can be generated from the reference application, and used to evaluate the results and execution times of the code, once processed by the FASTER toolchain.

3.2.4. Parallelism

A second version of the reference application will be delivered, based on parallel acceleration obtained using OpenCL and an accelerator such as a GPU. This application version will permit to evaluate the parallelism that the FASTER toolchain will be able to extract from the reference code.

Indeed, the raytracing application is, by nature, embarrassingly parallel. Each pixel, or even each sample in a pixel, can be computed independently, and by consequence in parallel. This level of parallelism can be parameterized, for example with the number of core available on a platform, or the number of threads that can be supported efficiently per core. Consequently, this criterion can be used to evaluate the efficiency of the toolchain, depending on the number of processing element or accelerator present at runtime.

3.2.5. Memory load

With parallelism comes the problem of concurrent memory accesses. The concurrent accesses to shared data, such as geometry information for example, can be measured and the efficiency of the platform can be determined.

On some systems, scene data may be replicated for each node for efficiency reasons. On others, concurrent accesses of a single, shared pool of memory gives satisfying results. The later approach has been used in the reference code, decomposing the image to render in tiles and spawning a thread of computation per tile using a shared scene description.

Consequently, the number of memory accesses, their average latencies and the general amount of memory can be potential evaluation criteria in order to judge the influence of the FASTER toolchain on the memory subsystem.

3.2.6. Extensions

The raytracer application is highly modular, and new components (i.e. new primitive types, new reflection models, new acceleration structures) can be added to extend its functionalities. The flexibility of the FASTER toolchain can be tested in expanding the capabilities of the raytracer, and measuring the additional workload involved. The run-time reconfiguration could also be tested in dynamically retargeting the acceleration resources of the platform according to the work load balance of a given scene (e.g. predominant primitive types).

3.3. HPC domain test case: Reverse Time Migration

Reverse Time Migration (RTM) is a high-end technique for generating images of the earth and is used by the oil and gas industry in complex geologies to give detailed subsurface images. Maxeler has provided an RTM application as an example of an HPC application for use by the FASTER project. Please see FASTER deliverable D1.1 for detailed information on the algorithm.

3.3.1. Baseline for Comparison

To evaluate the impact of the FASTER flow for RTM, we shall compare the FASTER demonstrator RTM application with an accelerated solution running without the benefit of FASTER partial reconfiguration.

The same FPGA hardware platform will be used for both the baseline application and the FASTER demonstrator, allowing the extra benefits of FASTER beyond those of the conventional FPGA implementation to be evaluated.

The likely hardware configuration is a Maxeler MPC-C500 compute node with 12 Intel Xeon CPU cores and 4 Virtex-6 FPGAs, where each FPGA is attached to 24GB of DDR3 RAM.

3.3.2. Overall runtime

The main evaluation criteria for a high performance Reverse Time Migration implementation is the time to run a job with a certain set of parameters. A full RTM job is made up of computing many independent shots and combining the individual shot images to produce a final output image; it typically contains a mix of compute and I/O.

A full RTM will typically run for several weeks on a large cluster, which is impractical to test. However a good proxy for overall runtime is to examine the runtime to compute a single shot image (four orders of magnitude less computation than the full result).

The use of the FASTER reconfiguration flow should reduce a shot's runtime, compared to the same hardware without the use of the FASTER flow.

3.3.3. Result quality

The second evaluation criterion is to produce a result of acceptable quality. This is a subjective measure, since the RTM output image is utilized by human interpreters.

It is not necessary for the FASTER demonstrator to give bit-identical results to either a non-accelerated (CPU) or non-reconfigurable FPGA version. However, providing an identical result to the non-reconfigurable FPGA version would be unambiguously acceptable.

If a non-identical result is achieved, the following evaluation criteria can be used:

1. Examine several 2D slices of the image at various points in the 3D volume.
2. Display the images with at least 256 levels of colour/greyscale. The colour scale should be clipped to the 99% level of the values histogram, to prevent a few very large values dominating the image.
3. Compare the difference between a reference image and the image produced by the accelerated computation, and examine the difference at a multiplication factor of x10 and x100.

Ideally differences between the two images should be mostly random and low intensity at the x10 multiplication factor.

Maxeler has developed a visualization tool called *Axion* (pictured below) which makes it easy to compare seismic data cubes such as the output images of RTM.

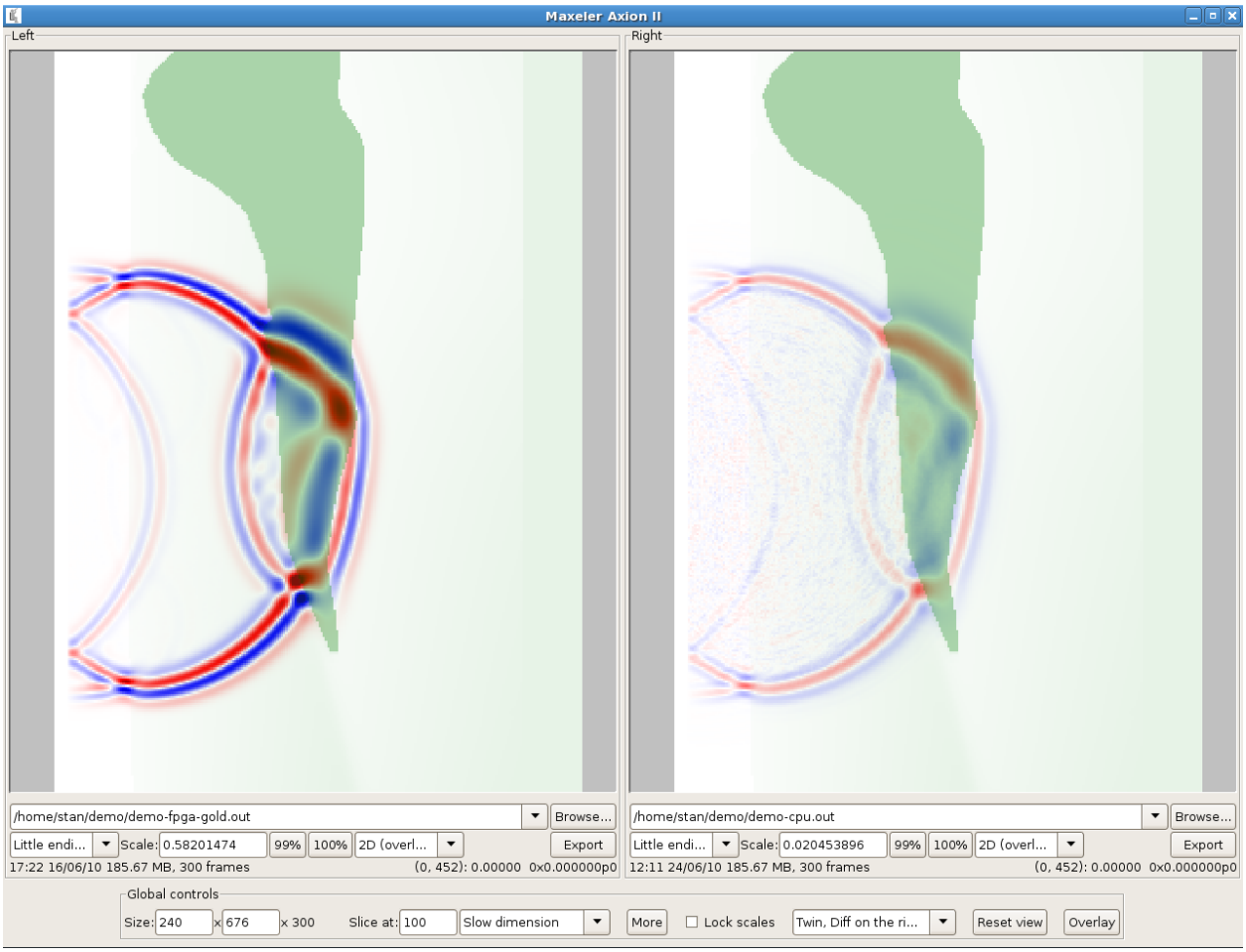

Figure 1 - Comparing two seismic wave datasets using the Maxeler Axion viewer. The left hand pane shows the original image, while the right pane shows the difference between the two data-sets (multiplied)

3.3.4. Capacity of hardware to support different parameter sets

Computing with FPGA dataflow engines is a spatial computation model where ultimately the amount of computation that can be performed is limited by the area available on the chip. This can impose limits on the ranges of parameter values that can be supported. Even when parameter values can be supported in theory, since the resources available to place & route FPGA bitstreams are limited, it may only be possible to optimally support a limited range of parameters.

For example, during RTM the *propagate* calculation runs for all timesteps, and the *imaging* timestep runs only every *N* timesteps. It is therefore necessary to determine what percentage of resources to allocate to the *propagate* calculation and what percentage to the *imaging* timestep. Parameters to decide this are not just the imaging step (*N*) but also the relative costs of the operations - for example the size of each stencil.

It is beneficial to be able to support the widest range of parameter values from a limited set of FPGA bitstream configurations. Ideally, the FASTER toolchain will enable more parameter values to be efficiently supported from a fixed set of FPGA bitstreams (or alternatively: a fixed amount of FPGA compilation CPU resource) than is possible without dynamic reconfiguration.

3.3.5. Contents of DRAM must be preserved during reconfiguration

The RTM application uses the FPGA's attached DRAM memories to store the seismic wavefields and earth model volumes during the computation of a shot. If the FPGA is partially reconfigured during the execution of a shot computation, it is important the DRAM contents is preserved to enable the computation to proceed with the correct data.

4. Conclusions

Three application domains are addressed by FASTER, from HPC to desktop applications and embedded systems. These comprise a wide range of very different requirements and give rise to different ways in which the results produced by the FASTER demonstrators will be evaluated.

We split the evaluation criteria we will examine into two categories: 'bottom-up' general evaluation criteria which apply to any implementation and

The general evaluation criteria will be able to be applied to every demonstrator:

- Dynamic Configuration Cost
- Area/size impact
- Clock frequency impact
- Performance
- Functional density
- Energy and power consumption
- Design effort
- Compile resources for primary bitstream and partial bitstreams

In addition, we will evaluate specific relevant characteristics for each individual demonstrator. For the NIDS application, this will include: update design effort and system down-time, overall area impact, performance (clock frequency and system throughput). For the raytracing application this will include: image quality, rendering speed, parallelism, memory load and extensibility. For the RTM application this will include: overall runtime, result quality, capacity to support parameter sets.

5. References

[1] FASTER Project, Deliverable D1.2 “Requirements of FASTER Models, Methods and Tools”, May 2012.

[2] FASTER Project, Deliverable D1.1 “Requirements of FASTER Applications”, February 2012.