Design Considerations of a Large Granule Dynamic Reconfigurable Fabric

Mua'ad M. Abu-Faraj

Abstract— Dynamically reconfigurable systems can provide wide-ranging support to processors as an adjunct to their usual Instruction Set Architectures. We describe the basic architectural features of an out-of-core dynamically reconfigurable fabric having predictive reconfiguration. The Out-Of-Core Reconfigurable Fabric (OOCRF) is intended for eventual extra-ISA support in a multicore architecture. In this paper we focus on the basic architectural structure of the fabric as it relates to a single core and give some simple examples of its use in (multi) register to (multi) register vector processing. In particular we introduce the notion of a composite configuration block and show how prefetching of configuration controls can be tied to standard instruction stream speculation. We also consider certain issues that can arise when a Dynamically Reconfigurable Fabric (DRF) is shared in a multicore system. We briefly describe three architectures supporting different multithreading models, and describe the role of authenticating locks for providing some form of hardware-based secured access to the DRF.

Keywords— Authenticating Locks, Dynamically Reconfigurable Fabric, Efficiency, and Configuration Caches.

I. Introduction

A Dynamically Reconfigurable Fabric (DRF) is a hardware device which can be configured dynamically to provide additional support to processors by providing a mechanism for implementing an algorithm in hardware. In this paper we are concerned with certain issues which arise when such a DRF is to be provided to a multicore system as an out-of-core service. We accept that we have control over the core's Instruction Set Architecture (ISA) and thus are able to specify a subset of the ISA which targets the DRF and configure any additional hardware support we deem necessary. We are interested in the design of such a DRF under a number of different circumstances, particularly as such a design would relate to the standard multithreading models (coarse, fine and simultaneous). Controlling access to the DRF is a particularly interesting problem, and in this paper we describe one such approach based on a simple authenticating lock mechanism.

Generally, the modern reconfigurable architecture emerged as a result of the development of the Field Programmable Gate Array (FPGAs), and they remain a popular choice to be used with processors in reconfigurable systems [1] because of their cheap price, flexibility, and programmability [2]. In [3], there is a description of various approaches to reconfigurable computing emphasizing the adaptability of these systems. However, the tradeoff between flexibility and functionality usually indicates that the improvements are not as high as they can be in Application-Specific Instruction Set Processors (ASIPs) [4] or Application-Specific Integrated Circuits (ASICs) [5].

M. M. Abu-Faraj is with the Department of Computer Information Systems, The University of Jordan, Aqaba, 77110 Jordan (e-mail: m.abufaraj@ ju.edu.jo).

A discussion of the comparative pros and cons between the general purpose CPU, the ASIP or ASIC and the intermediate DRF can be found in [4], [6]. Clearly, there is a cost-performance tradeoff in deploying a DRF in place of (e.g.) an ASIC, since it likely that the ASIC will be faster for any given algorithm than the DRF and will likely have a lower gate count. Formalizing a cost/efficiency metric for DRFs is itself an interesting area and one we turn to (albeit very briefly) at the end of this paper.

Our concern in paper is with the exposition of the initial architectural issues subtending an out-of-core dynamically reconfigurable fabric (OOCRF) which will ultimately be shared by a number of risc (or post-risc) cores (although here we ignore all multicore issues). The intention is to provide dynamically-reconfigured assistance to the cores, which we take to be OOO in design, having a simple core ISA along the lines of an Open RISC core [7] with additional simple extensions to this ISA in support of the OOCRF. The OOCRF itself we envisage as being essentially a load/store architecture based around vectorial register-register functions i.e., such that OOCRF supports computations of the form $R_{out} = f^i R_{in}$ where R_{out} and R_{in} are vectorial registers and the f^i belong to a restricted class of operations. Specifically, we do not perceive the OOCRF as being an essentially general-purpose dynamically reconfigurable fabric (and therefore the OOCRF lacks some of the more general features of the FPGA). Instead, two issues motivate us: the role of the OOCRF in multimedia, imaging, visualization and simulation with (e.g.) specific interest in quasi-DSP applications, and the desire to maximize configuration efficiency E_C.

II. Related Work

Dynamically reconfigurable architectures have a long history both as stand-alone alternatives to classical static architectures or as assistive architectures [8], [9] and, as befits the maturity of the subject, have a long and impressive literature which is too large to be other than sampled here, and so we restrict our attention to three representative examples which have a bearing on this work.

Interest in dynamically reconfigurable systems has been renewed with the advent of large-scale reconfigurable FPGAs and we now see a variety of reasons for their deployment including such diverse applications as array processors for wireless broadband technologies [10], cryptography [11] and more embedded applications (such as automotive applications [12]).

Reconfigurable computing has found applications in a wide variety of areas [13] and as a result has a large literature dating back over decades. It is impossible to cover the field adequately in a paper of this size, and so we restrict our attention to three representative examples, which have a bearing on this work.

Chimaera [14] is an example of a reconfigurable system that works together with a host processor as a tightly coupled unit, with direct access to its register file in order to decrease communication time. The reconfigurable fabric consists of FPGA logic designed to support high-performance computations. Hence, its reconfiguration granularity is fine. The reconfigurable array is connected to a MIPS R4000 processor. In [15] GARP was introduced as a reconfigurable system attached to a MIPS II processor as a co- processor. It can also be considered as a loosely coupled Reconfigurable System. GARP also uses FPGA logic for the reconfigurable fabric, so it can again be considered as a fine grained reconfigurable array. REMARC [16] consists of a reconfigurable unit coupled to a MIPS II ISA based RISC machine. REMARC is arguably a loosely coupled reconfigurable architecture. We began our design based on the REMARC array structure, although we rapidly diverged from the REMARC system.

III. Core Microarchitecture overview

The core microarchitecture is shown in Fig. 1 below:

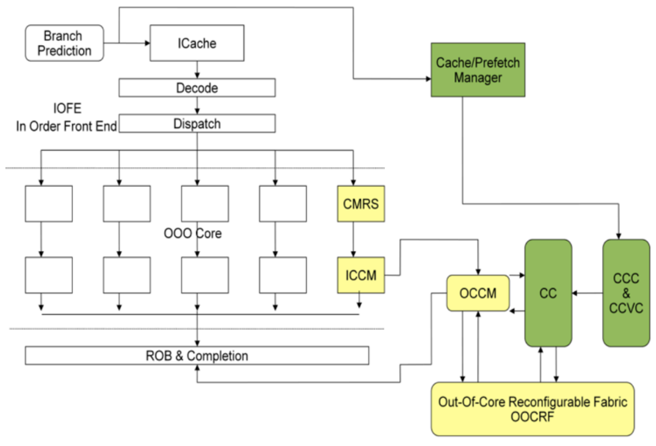

Fig. 1 Core Microarchitecture

The architecture is schematically shown as a standard Out-Of-Core (OOO) superscalar core [17], [18] with an in-order front end and completion/retirement under control of a reorder buffer. Two components associated with the OOCRF are shown in core (and should be considered as conventionally dispatchable functional units within the core). The CMRS (Configuration Manager Reservation Station) and ICCM (In-Core Configuration Manager) present the OOCRF to the core, and in some senses act as an in-core proxy for the OOCRF. As shown later, the CMRS/ICCM pair are effectively simplified load/store units, treating the OOCRF almost as an independent component of the memory hierarchy. Interactions between the OOCRF and the in-core ICCM as well as the ROB/Completion manager are mediated by the OCCM (Out-Of Core Configuration Control Manager) (see below). Configuration control is driven into the OOCRF via the Configuration Controller (CC) which sets up the data pathways in the OOCRF corresponding to the current configured function f^i. Finally, in the instruction fetch pathway we have a configuration prediction mechanism attached to the standard fetch pathway (Configuration/Prefetch Manager) which attempts to prefetch configuration control data for caching into the Configuration Control Cache which is in turn associated with a small Configuration Control Victim Cache (CCVC).

IV. OOCRF Instruction Set Architecture

Interactions with the OOCRF are fundamentally those of a load/store architecture (carrying conventional order constraints). The core ISA sees the OOCRF as having three ports: a set of in vector registers, a set of out vector registers and a configuration control block (CCB) which is a data structure defining both the configuration and timing. (Although all examples configuration functions we show below are combinatorial, there is no reason to so restrict the OOCRF).

Under the current architecture, the data ports to the OOCRF consist of two unidirectional vector register sets (srcVRF and dstVRF) as shown in Fig. 2.

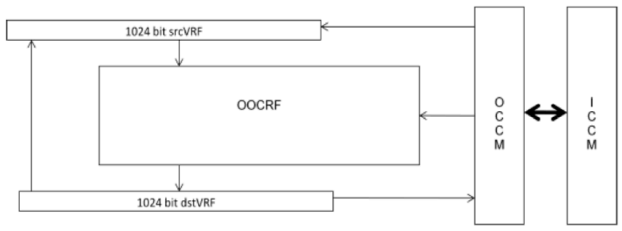

Fig. 2 OOCRF Register Architecture

Depending on the configuration set into the OOCRF, the vector register file(s) appear as any (restricted) partition of 1024 (bits) relative to the elements {128,64,32,16,8} which correspond to the data types complex double (C^D), complex single/float double/int double ($C^S F^D I^D$), float single/integer single ($F^S\ I^S$), halfword fixed point (fP^H) and byte (B). The vector register files therefore correspond to a (full) data transfer of 128 bytes, and we expect that vector data will be cache block aligned, even when the cache line is less than 128 bytes. Data transfer from the memory hierarchy or processor register files into or out of the vector registers is controlled by the ICCM coordinating with the OCCM.

Equivalently, configuration control data must be transferred from memory to the OOCRF CC (Configuration Controller) via the OCCM. The situation here is somewhat different from the straightforward vector register accesses, in that configuration control data is cached by the

Configuration Control Cache (CCC) supported by Configuration Control Victim Cache (CCVC) (see section V below).

Notice that there is a direct path from the dstVRF to the srcVRF. In a number of DSP applications we have been exploring, we have noticed a number of occasions in which consecutive functions apply to consecutive outputs i.e., the frequent occurrence of composite functions of the form

$$w = f^n(f^{n-1}(\dots f^1(v)\dots)) \quad (1)$$

leading us to include the pushback instruction dstTOsrc below.

These considerations lead to a relatively straightforward extension to the core ISA, with effectively eight new instructions (there are error conditions not discussed here) of the generic form:

LDCCM CCB^i
LDCpCCM $CCB^{1\ of\ n}$
LDsrcVRF M_p*Address*
MVsrcVRF *RFile*
ExCfg
STdstVRF M_p*Address*
MVdstVRF *RFile*
dstTOsrc

--corresponding to the actions of loading the configuration defined by the i^th CCB (memory → CCM), loading the first block of a composite CCB (see Section V below), loading the source vector register file (mem→srcVRF), moving (copying) core register data to the srcVRF (RFile→VRF), enabling execution in the OOCRF, the respective dstVRF store (memory) and copy (core registers) and the dstVRF to srcVRF operation described above.

V. Configuration Caches and Speculation

A central thesis of our approach is that the OOCRF fabric exhibit coarse-grained reconfiguration i.e. the reconfigurable components are at the level of (relatively) sophisticated functional units (floating point adders/multipliers, integer adders/multipliers) rather than the fine-grained reconfiguration (gates/CLBs) that can be found in FPGAs. Thus, although the total gate count in the OOCRF is expected to be quite high, configuration control (i.e., the designation of a functional path from the srcVRF to the dstVRF via the OOCRF) essentially boils down to data steering in 8,16,32 and 64 bit quantities (the 128-bit complex double data type being only 2 64-bit double floating point data types with complex arithmetic explicit in the configuration control). Depending on the density of the OOCRF and its component makeup, the data steering size of the configuration control can be quite small. Although not quite fully correct, data steering configuration control can be thought of as that assignment of multiplexor steering bits which enables the construction of a (possibly sequential) path from srcVRF to dstVRF in the OOCRF. With this in mind, a configuration control block (CCB), consisting of the data steering block (DSB) and associated finite state machine block (FSMB) are small relative to fine-grained reconfigurable systems.

The notion that the reconfiguration fabric is coarse-grained rather than fine-grained (FPGA- like) has a significant impact on the strategy we use to prefetch and cache configurations. Our situation is considerably simpler than the caching and prefetching models described by Hauck and Li and colleagues [19], [20], [21] which are suited for more general FPGA environments. Specifically, unlike the Hauck-Li models, our configuration control blocks (CCBs) are constant in size, and therefore suffer no variation in load latency beyond that caused by the memory hierarchy itself. Similarly, since our reconfigurable fabric is coarse-grained rather than fine-grained, our CCBs (depending on the coarse-grained fabric we are considering) subtend few cache lines. Moreover, we do not use the reconfigurable fabric itself to store the configurations; since our fabric is not that of an FPGA, actual reconfiguration time is not as significant as it is in the FPGA case. While we admire the prefetching techniques described in [21], both the static model (with explicit prefetch instructions) and the dynamic models (pure and hybrid) are overly complex for our requirements. In fact, we believe we can tie configuration prefetching (and caching) to the standard instruction fetch core, and thereby greatly simplify the associated prefetch hardware (which is already mostly present in the IF stage of the core pipeline). Finally, we note that since we prefetch to a configuration cache and not to the configuration fabric itself, a prefetch error for us is more akin to a data prefetch error than an instruction stream prefetch error.

CCBs are therefore cached in the Configuration Control Cache (CCC) and its associated victim cache (CCVC), with the intention of prefetching CCBs preparatory to their use in the OOCRF. Although data prefetching is important in vectorial systems, it transpires that in composite vectorial processes of the form $w = f^n(f^{n-1}(\dots f^1(v)\dots))$ given previously, VRF register definition (load) from the core is the exceptional load while VFR register definition from the OOCRF destination registers is the more common srcVFR load. Under these circumstances, the most costly miss is not the data miss, but the miss on the required CCB for the k'th function f^k.

As shown in Fig. 1, we have tied configuration prefetching to the instruction stream speculation mechanisms. To do this, we introduce the concept of the composite CCB which is simply the CCB described above together with a linking data structure from CCB I to CCB i+1 in the composite chain. Prefetching then reduces to attempting to ensure that as much of the composite CCB is cached prior to any repeated execution of the first functional configuration in the composite sequence. One way to do this is to speculate on the prefetching of the composite chain by associating composite chains with the basic block that each composite chain appears in. We note of course that configuration speculation is more closely akin to data prefetching and, being non-binding, is only a performance issue if the speculation is incorrect. Thus the OCCM has no need of any form of speculative recovery process.

As shown below in Fig. 3, the configuration prefetch mechanism is tied to a conventional branch predictor, which (in the case given) is a modified Yeh-Patt 2-level predictor [22].

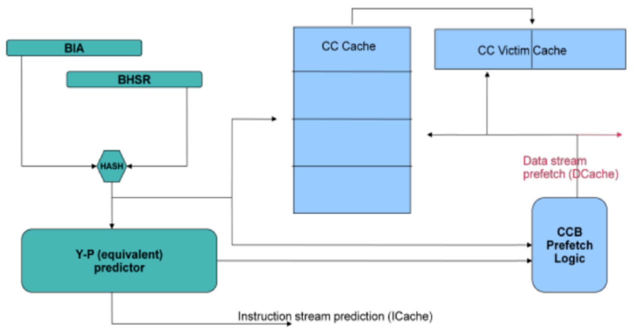
BIA
BHSR
HASH
CC Cache
CC Victim Cache
Data stream prefetch (DCache)
Y-P (equivalent) predictor
CCB Prefetch Logic
Instruction stream prediction (ICache)

Fig. 3 Configuration Speculation

VI. General Architecture Considerations

As indicated previously, we are concerned with sharing a DRF in a multicore system. In fact, the actual structure of the DRF (in terms of its core reconfigurable components) is of little interest to us (at least for this paper); we are not concerned if the basic reconfigurable entities are at the level of the FPGA logic block or at the level of a REMARC-like subprocessor. Perhaps the one area where the structure of the DRF could become important would be in the relationship of the DRF to the multithreading model the DRF best supports. We argue (but concede opposing viewpoints) that there might be three DRF architectures which would best suit each of the coarse grained, fine grained and simultaneous multithreading models. In a coarse-grained environment, it is not unreasonable to assume that configurations swap in and out of the DRF at about the granularity of the thread swaps, and so a coarse- grained DRF might be deployed in multicycle activity (e.g., in a functional transform such as a wavelet transform). On the other hand, a fine-grained core architecture might expect results from the DRF during the fine-grained quanta, and so the DRF might be expected to host algorithms which terminate in a few cycles at most (e.g., perhaps a round of an encryption algorithm such as AES). By the same token, cores supporting simultaneous multithreading might expect the DRF to be able to return results within one or two cycles, and so the DRF functionality might be more suited to simple ISA extensions (e.g., an instruction to swap bytes and XOR the swapped lower byte with a mask). To this end we propose three increasingly complex models for the DRF, each of which is a superset of the preceding model in the sense that anything the preceding model could accomplish could be accomplished by the succeeding model but at greater cost and lower efficiency (see Section VIII). In this paper, we are concerned principally with the sharing mechanism as it relates to these three models. We began our models with a simple example of an Out-Of-Core DRF supporting a single core [23].

The position of the DRF relative to the processor dictates to a large extent the protocols for accessing the DRF in a shared environment. Evidently, an in-core DRF (i.e., a DRF designed as an in-core functional unit) is not sharable between cores. At the other extreme, one could place the DRF out in the I/O space, to be configured and have data written to and read from it as if it were an I/O device (e.g., as an FPGA might get attached). Sharing protocols (access to all or part of the DRF) could then be devolved to the operating system, and rogue access could be inhibited in the usual way (e.g., by requiring the processor be in a supervisor mode). The drawback here is that long access times would be involved in accessing the DRF via the operating system which mitigates against its use as an ISA extension tool. (Evidently, if the DRF is conceived as supporting very long- haul processes (hundreds to thousands of cycles, such as supporting massive data encryption) then the cost of routing via the operating system no longer counts). On the other hand, if access to the DRF at (or near) single-cycle processor core speeds is desired, then the DRF must be brought closer to the processor cores and outwith the control of the operating system. In a sense, the DRF can be thought of as having a position relative to the cores as a shared (L2/L3) cache might have (although they are different architectural entities). We refer to such a DRF as an Out-Of- Reconfigurable Fabric (OOCRF), and an example of such a design for a single- core system can be found in [23]. This now raises the issue of access protocols to the DRF which must be implemented by the DRF itself.

The reason we propose all three models below (rather than a single encompassing model) is driven by the design goals of the cores together with the cost/efficiency of the DRF itself. Every gate in the DRF *not* dedicated to computation (i.e., every gate in the DRF dedicated solely to the provision of dynamic reconfiguration or to (e.g.) access controls) is a gate which reduces the overall efficiency of the DRF relative to a single dedicated circuit performing the same task (see Section VIII below). If the cores are designed as coarse-grained multithreading systems, the increased cost and lower efficiency of a fabric which can support simultaneous multithreading is unwarranted. An example of a DRF functional block is given in Fig 4 below. Again we stress that the actual design of such a block is of less importance to us than the issues subtending the sharing of such blocks, but the simple model shown in Fig. 4 allows us to concretize certain aspects of the sharing models of Section VII. We see the basic sequence involved in using a DRF (or part thereof) as follows:

- Obtain permission to access (all or part of) the DRF;
- Configure (all or part of) the DRF access has been granted to;
- Load data into the DRF;
- Issue an ENABLE command;
- Read data from the DRF on completion of the algorithm;
- Unlock that part of the DRF permission had been granted for.

Throughout the above steps, we assume (see Section VII) that access to the DRF is gated based on appropriate permission having been granted. We now look at the above steps (other than obtaining access rights) in the light of Fig. 4. *Configuring* the DRF means setting the various data and control pathways to accomplish the specific task requested. The architecture which supports this is not shown in Fig. 4. I/O is done via uniquely addressable *staging registers* which are available at both the input and output. The relationship between registers and components of the DRF can be steered using an input (output) steering network. We are not concerned with how a user of the DRF identifies that a process loaded into the DRF has terminated; however, one simple mechanism for determining if an n-cycle process has completed would be the provision of read/writable count down register decremented every clock cycle by the DRF. The user would be expected to know (and load) the correct number of cycles.

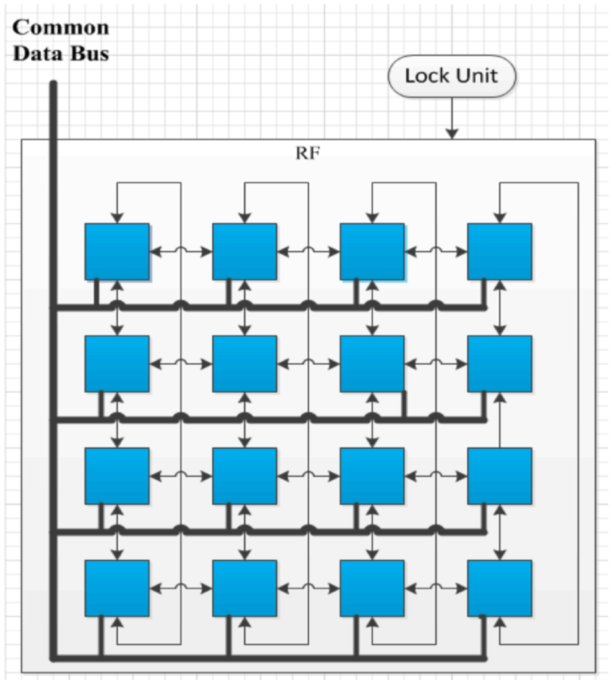

Fig. 4 Toroidally Connected DR Array with Common Data Bus

VII. Models

We consider three scenarios; where the cores are designed to support coarse-grained multithreading, where they are designed to support fine-grained multithreading, and where they are designed to support simultaneous multithreading. In all cases, we need to prevent collisions in the DRF either from multiple threads under the same context or from threads arising in different contexts. Referring to the actions in Section VI above, we see that configuration, data loading and unloading, enabling and unlocking are all areas where collisions can occur. Since no OS support can be provided, the DRF must provide at least basic support to avoid such collisions.

Central to all models is the notion of the *authenticating lock*. This is a hardware construct constructed within the *Locking Manager* of the DRF with explicit ISA support in the cores themselves. An authenticating lock is essentially a multivalued semaphore where the *granted* value (here called ID) is a unique ID assigned by the Locking Manager if access to the requested resource(s) within the DRF has been granted. The ISA (and the DRF) must support an atomic primitive of the form:

RequestLock DRF_Resource, ID

where DRF_Resource is a DRF Resource identifier (or pointer to a data structure containing a list of such resources – see models 2A and 2B below). ID is a *random* integer returned by the Locking Manager to the caller (generated in our models by a Linear Feedback Shift Register(s) (LSFR) within the Locking Manager). A sufficiently large LFSR (32 bits) is still (gate wise) relatively inexpensive and although its use (below) is not secure in any strict sense of that word, it would still require an assailing process determined to circumvent standard access protocols to the DRF to attempt to identify the ID within a (relatively) large space of such IDs (2^n-1 for an n-bit maximum LSFR). Once an ID has been granted, all subsequent accesses to the DRF *must* be gated by the Locking Manager which checks the gating ID against the ID reported by the requesting processors. Thus (for example), loading a configuration into the DRF (in whole or in part) involves an instruction of the form:

LoadConfiguration,Configuration,DRF_Resource, ID

where ID is the same ID provided to the process by the DRF.

For simplicity, we assume that the DRF consists of a toroidally-connected array of elements of the form given in Fig. 4. I/O to/from the array elements is gated via a Common Data Bus, while inter-element connectivity is permitted by cardinal-neighbor connections. As holds for the standard I/O, inter-element I/O can only be set up provided the Lock Manager has released the appropriate locks to the requesting process.

A. Model 1

Model 1 is the simplest model, where single cores have total access to the DRF at any one time, as shown in Fig. 4. Here, a single global lock controls all access to the entire DRF. This model is best suited for coarse-grained multithreading, which means only one processor has access to the DRF. Thus our assumption for this model is that one processor (thread context, see below) has total access to the DRF for the duration of a coarse-grained swap quantum (see Section VI). While this model allows one processor to access the DRF, it can be simply extended to permit multiple threads from a single core to access the DRF under a [<lock by core> <lock by thread>*] protocol. The Lock Manager here merely has to track one lock for the entire system, and is thus simpler and cheaper than the Lock Managers in the remaining two models below.

B. Model 2A

Models 2A and 2B differ only in their degree of complexity relative to the Locking Manager. The architectural argument for the distinction between them relates again to the threading models; in case 2A, we assume fine-grained multithreading, and thus fine-grained use of the DRF. This leads us to assume that the DRF should be partitionable into large subsets to accommodate the extent of DRF resources that a process completing in 10s of cycles might need. In model 2B, we assume simultaneous multithreading, and thus we expect processes in the DRF to execute only for a few cycles at most and so we permit even finer partitioning of the DRF. Otherwise the models coincide. It will be clear at the end that Model 1$\subset$ Model 2A $\subset$ Model 2B. Fig. 2 (Model 2A) shows a DRF shared between 4 different processes (thread contexts) with the Lock Manager controlling access on a per-row basis.

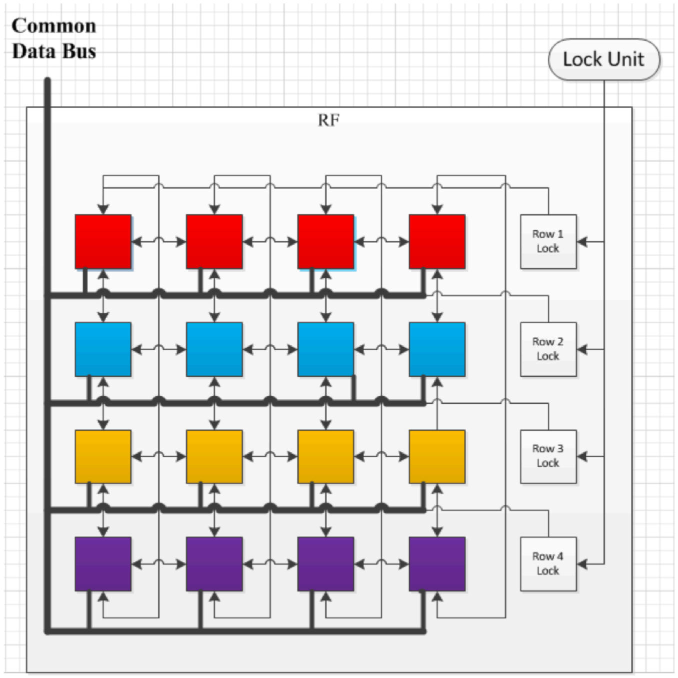

Fig. 5 Model 2A

C. Model 2B

Fig. 6 shows the architecture of model 2B. Here, the partitioning granularity and locking granularity are at their finest, with concomitant increases in complexity and cost and decrease in efficiency. In this model the processor locks only the exact number of logic blocks it needs to execute the application assigned to it. Here, each logic block has an associated lock (and so the locking is at the finest granularity possible) and processors can request either specific groups of logic blocks, or (with significant increased complexity) specific patterns of grouping of logic blocks. To see the difference, note that a specific grouping might be "blocks (1,1) (1,2), and (1,3)" while a specific pattern might be "any three neighbor-connected blocks in a row".

Architecture 2B raises (in general) the interesting possibility of *dynamic re-assignment* within the DRF i.e., the re-mapping of assigned subsets of blocks to new assigned subsets of blocks in such a way as to preserve the requested communication topology of the original assignment but in addition permitting a new assignment to be layered into the DRF. This is (generally) a difficult problem related to the graph isomorphism problem.

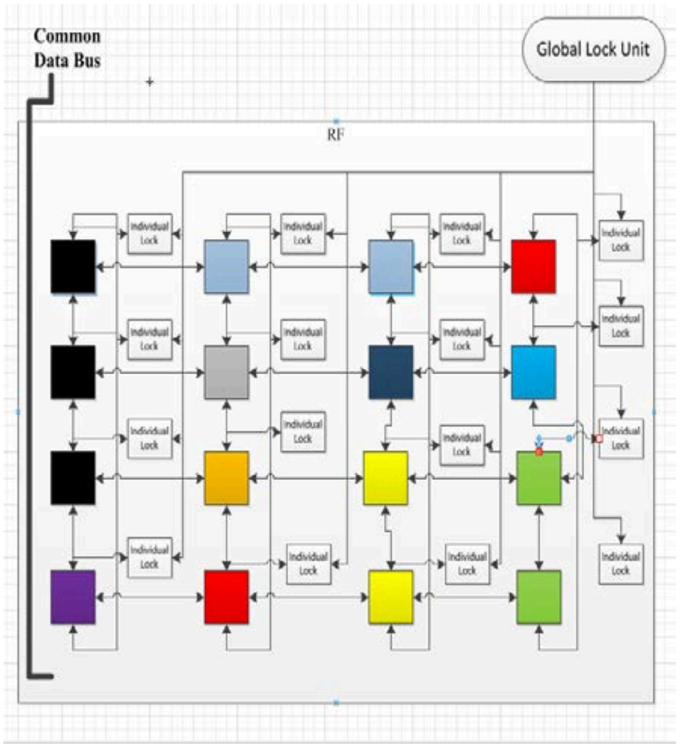

Fig. 6 Model 2B

VIII. Efficiency Metrics

Usually, the performance of a dynamically reconfigurable system is measured against a particular algorithm or a suite of such algorithms which have been implemented on the DRF. Thus, one might compare the speed with which Algorithm A runs to completion on a CPU (ASIC/APIC) against the DRF. This does not serve to justify our argument that different models of a DRF might be appropriate for different multithreading architectures, and so we refer to a simple efficiency measure for DRFs we introduced in [23]. Here, the efficiency E_C, can be defined as follows:

$$E_C = \frac{G_F}{G_F + G_R} \qquad (2)$$

where G_F is the gate count associated with functional components in the DRF and G_R is the gate count associated with components in the DRF whose only role is enabling reconfiguration. Clearly, E_C relates the overall cost of the DRF against its purely support infrastructure. When the granularity of the DRF elements is fine (on the order of a few gates/logic blocks) increasing the partitionable complexity of the DRF (Model 2B) can have catastrophic effects on the efficiency. For an $n \times n$ array-connected DRF as shown in Fig. 4, Fig. 5, and Fig. 6, the partition complexity has order $O(1)$, $O(n)$, and $O(n^2)$ respectively for models 1, 2A and 2B, and so the impact on the efficiency above varies as $O(1)$, $O(n)$, and $O(n^2)$ respectively. For constant G_F=10 and G_R=5, this effect is shown in Fig. 7. Note that Models 1 and 2A are asymptotically equivalent, while Model 2B has a constant efficiency (losing the gains accrued by Models 1 and 2A by virtue of its quadratic partition complexity).

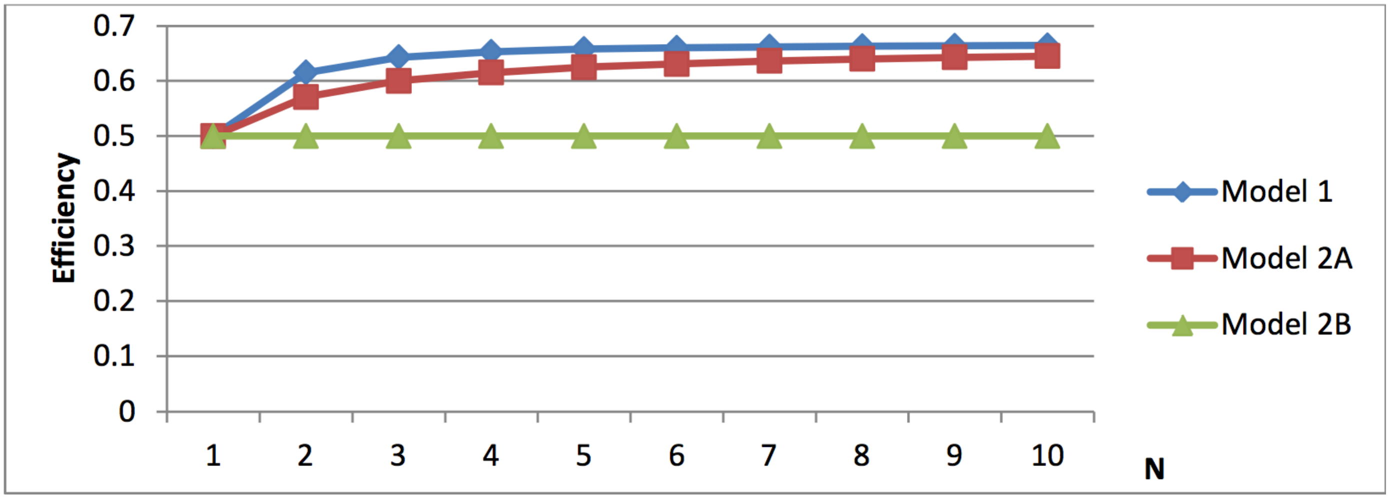

Fig. 7 Efficiency for the 3 Models

IX. Conclusion

In this paper we have described the basic architectural structure of an out-of-core reconfigurable fabric organized at a coarse-grained level of configuration granularity interacting with the OOO core as a load-store architecture. Critical parts of the architecture are motivated by the observation that in certain DSP applications for which the reconfigurable fabric was initially considered the sequence of configuration events followed a composite rule. Thus the architecture supports direct remapping of its destination to source vector registers, and in particular, the notion of composite configuration block prefetching tied to instruction speculation is a direct result of this observation. We have also demonstrated 3 DRF architectures suitable for shared deployment in a multicore system and shown how a simple access protocol for these architectures can be provided using an authenticating lock. We justify the architectural models by appealing to a simple efficiency metric.

References

[1] R. D. Wittig, "OneChip: An FPGA Processor With Reconfigurable Logic," in *IEEE Symposium on FPGAs for Custom Computing Machines*, 1995, pp. 126-135.
[2] M. B. Gokhale and P. S. Graham, *Reconfigurable Computing: Accelerating Computation with Field- Programmable Gate Arrays*: Springer, 2005.
[3] L. Carro and A. Fi, *Dynamic Reconfigurable Architectures and Transparent Optimization Techniques*: Springer, 2010.
[4] M. Platzner and N. Wehn, *Dynamically Reconfigurable Systems: Architectures, Design Methods and Applications*: Springer, 2010.
[5] M. Smith, *Application-Specific Integrated Circuits*: Addison-Wesley Professional, 1997.
[6] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Comput. Surv.*, vol. 34, pp. 171-210, 2002.
[7] Open Risc, http://opencores.org/openrisc (retrieved October 2013).
[8] I. R. Greenshields, *A Dynamically Reconfigurable Vector-Slice Processor,* Proc. IEE Part E, 129(5), pp. 207-215, 1982.
[9] I. R. Greenshields, *A Dynamically Reconfigurable Multimodal Architecture for Image Processing* in Parallel Processing for Computer Vision and Display, eds. P.M. Dew, R.A. Earnshaw and T.R. Heywood, Addison-Wesley, Reading MA, pp. 153-165, 1989.
[10] W. Han, Y. Yi, M. Muir, I. Nousias, T. Arslan, and A.T. Erdogan, *Multicore Architectures With Dynamically Reconfigurable Array Processors for Wireless Broadband Technologies*, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, Vol. 28 No. 12 pp. 1830-1843, 2009.
[11] I. Damaj, M. Itani, and H. Diab, *Serpent Cryptography on Static and Dynamic Reconfigurable Hardware,* Proc. Of the IEEE Int. Conf. on Computer Systems and Applications pp. 680-684, 2006.
[12] J. Becker, M. Hubner, G. Hettich, R. Constapel, J. Eisenmann, and J. Lucka, *Dynamic and Partial FPGA Exploitation*, Proc. IEEE Vol. 95, No. 2, pp.438—452, 2007.
[13] P.-A. Hsiung, M. D. Santambrogio, and C.-H. Huang, *Reconfigurable System Design and Verification*: CRC Press, 2009.
[14] S. Hauck, T. W. Fry, M. M. Hosler, and J. P. Kao, "The Chimaera reconfigurable functional unit," in *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 87-96,1997.
[15] J. R. Hauser and J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor," in *IEEE Symposium on FPGAs for Custom Computing Machines*, 1997, pp. 12 - 21.
[16] T. Miyamori and K. Olukotun, "REMARC:Reconfigurable Multimedia Array Coprocessor," *IEICE Transactions on Information and Systems E82-D,* pp. 389-397, 1998.
[17] J. P. Shen, and M. Lipasti, *Modern Processor Design*, McGraw-Hill, New York NY, 2005.
[18] J. L. Hennessy, J.L. and D.A. Patterson, *Computer Architecture A Quantitative Approach (Fourth Edition)*, Elsevier Morgan Kaufmann, San Francisco CA, 2007.
[19] S. Hauck, *Configuration prefetch for single context reconfigurable processors,* Proc. 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays pp. 65-74, 1998.
[20] Z. Li, K. Compton, and S. Hauck, *Configuration caching management techniques for reconfigurable computing,* IEEE Symposium on Field-Programmable Custom Computing Machines, Napa Valley, CA, pp.22-36, 2000.
[21] Z. Li, and S. Hauck, *Configuration prefetching techniques for partial reconfigurable coprocessor with relocation and defragmentation,* Proc. 2002 ACM/SIGDA Tenth Int. Symposium on Field-Programmable gate arrays, pp. 187- 195, 2002.
[22] Yeh, T.Y. and Y.N. Patt, *Two-level adaptive training branch prediction*, Proc. 24'th Annual Int. Symposium on Microarchitecture, pp. 51-61, 1991.
[23] M. Abu-Faraj and I. Greenshields, "Architectural Considerations for an Out-Of-Core Dynamically Reconfigurable Fabric," in *1st Int'l Conference on Advanced Computing and Communications*, pp. 52-57,2010.

Mua'ad M. Abu-Faraj received the B.Eng. degree in computer engineering from Mu'tah University, Mu'tah, Jordan, in 2004, the M.Sc. degree in computer and network engineering from Sheffield Hallam University, Sheffield, UK, in 2005, and the M.Sc. and Ph.D. degrees in computer science and engineering from the University of Connecticut, Storrs, Connecticut, USA, in 2012.

He is, at present, assistant professor at the University of Jordan, Aqaba, Jordan. He is currently serving as reviewer for the IEEE Micro, IEEE Transactions on Computers, Journal of Supercomputing, and International Journal of Computers and Their Applications (IJCA). His research interests include computer architecture, reconfigurable hardware, cryptography, and wireless networking.

Dr. Abu-Faraj is a member of the IEEE, ISCA (International Society of Computers and their Applications), and JEA (Jordan Engineers Association).