Reconfigurable Computing Cluster: three domains architecture solution

Francisco Sanchez, Julio Dondo, Fernando Rincon, Francisco Moya, Juan Lopez
School of Computer Engineering, University of Castilla-La Mancha
Ciudad Real, Spain
{Francisco.SMolina, JulioDaniel.Dondo, Fernando.Rincon, Francisco.Moya, JuanCarlos.Lopez }@uclm.es

***Abstract*—The incorporation of reconfigurable devices (FPGAs) to traditional grids of processing units such as GPUs add more complexity to the design and exploitation of heterogeneous reconfigurable computing systems. A lot of proposal were made to facilitate critical aspects such as network interconnection, devices access, programming models... but a few proposal offers a complete flexible solution. For this purpose, we need to cover three different domains: the user access (services), the application developer (programming models), and finally the system developer architecture (architectural model). In this way this work proposes a complete dynamic reconfigurable computing system integrating aspects to offer a three domain architecture solution.**

***Index Terms*—FPGA, Dynamically reconfigurable, Reconfigurable Computing, Programming Models, Architectural Models.**

I. Introduction

The multi-node dynamically reconfigurable distributed computing compromise three different domains (fig. 1) related to type of system access:

- Architecture development domain
- Application development domain
- User access domain

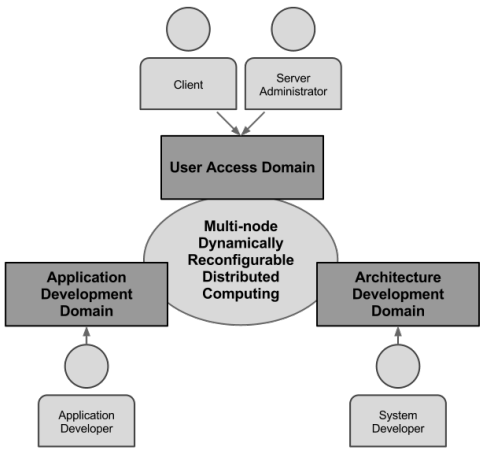

Fig. 1. Reconfigurable computing: three domains

Architecture development domain is related to the design of reconfigurable computer systems, including, for example, the integration of computing devices, how to interconnect them or the amount of involved resources. The main concerns are about topology, networking, memory, bandwidth, or computing devices models.

Application development domain deals with the application development process, from design to implementation and testing. In this domain, the main concepts involved are programming languages, tools, programming models, and workflows.

Finally, *user access domain* is related to the interaction of the final user with the reconfigurable computing system. Users are those requiring computational capacities for application deployment, and those providing platform administration roles. The user submits jobs and requires a variety of services to easily run their applications in a safe and successful way. On the other side, grid service provider requires services for managing system state, or to perform the administrator role. The main concepts involved in this domain are: a) from client side: application deployment, user access interface, resource demand; and b) from administrator side: performance, management tools for resource allocation and scheduling, security policy, etc.

Despite its powerful capabilities, current heterogeneous reconfigurable computing platform does not offer an integral flexible solution involving the three domains detailed previously. A proposal integrating solutions for these domains is urgently required. This solution must also posses the capacity to offer scalability and flexibility in order to cope with the growth of computational resources for *High Performance Computing* (HPC) demands. These characteristics include aspects such as facility to integrate different technologies and *High Performance Reconfigurable Computing* (HPRC) architectures, facility to provide resources access transparency to users and so on.

In this way, we propose a complete architecture that offers solutions in these three domains. The propose can be summarized in a reconfigurable computational architecture that simplifies the system development, offers different programming models and provides clusters services to exploit it.

II. Related work

The integration in distributed reconfigurable computing of those three domains defined in section I, have been made partially in HPRC, where several reconfigurable devices (FPGAs)

are offered with general purpose processors beside a set of libraries facilitating access and accessibility of resources.

High Performance, flexibility and low power consumption are the main reasons to integrate FPGAs also in HPC [1].

Most of the solutions in this concern place FPGAs as a simple coprocessor of a master entity (i.e. an on-board CPU) that typically runs a control program. FPGAs are, thus, relegated to a lower level, behind the processor. This is the dominant role of FPGAs both in High Performance Embedded Systems [2], [3], [4], [5], [6] and HPC Servers.

Examples of this configuration are the SGI Altix servers [7], Netezza for data warehouse applications [8], the Convey HC1 and HC-1ex hybrid computers [9] or the Cray XD1 [10], just to name a few of them. In some cases, FPGA technology is hidden behind an extended instruction where designated operations are accelerated in the hardware fabric. In other cases, FPGAs can only be accessed through a tightly coupled processor using a closed API.

Few commercial products fall out of this group, such as the RYVIERA and COPACABAN platforms. SciEngines [11] provides the developers with a bare reconfigurable platform with several FPGAs blades at the same level than processors. However, the development environment is not trivial for non-expert hardware personnel.

Most of works using FPGA for HPC repeat the strategy of integrating an accelerator into applications to speedup the execution of the kernel of an algorithm [12] [13]. Nevertheless, this strategy is not intended to execute the whole application in hardware. The approach presented in [14] represents an evolution with respect to the acceleration of a single algorithm. It offers an architecture where reprogrammable hardware resources can be used as if they were resources managed by the operating system, abstracting in this way user applications. The proposed architecture is based on a card with partially reconfigurable FPGAs connected to the bus of a general purpose computer. This architecture offers a system to facilitate loading those hardware components needed to accelerate the application, through a software layer that incorporates these FPGAs as if they were additional system resources. This work does not provide hardware communication transparency and replication services.

A. Contributions to development models and tools

To assist in the definition and building process of the hardware components, designers have relied in the use of *High Level Synthesis* (HLS) tools. Some approaches have been made such as Impulse-C [15], Handel-C [16], and Transmogrifier C [17]. Lately, important efforts have been done considering only a subset of particular high-level languages (normally C/C++) for HDL translation such as Mentor Graphics' Catapult C [18], Synfora's PICO (now Synopsis) and AutoESL's AutoPilot (now Xilinx). They have been able to achieve a level of resource-usage efficiency comparable to that obtained using hand-written RTL code [19]. Another interesting initiative that takes advantage of HLS tools applied to HPC related problems is the recent announcement of Altera [20] that uses OpenCL as the unique programming model for FPGAs, GPUs and CPUs. In the same line (OpenCL synthesis capabilities) can be found the project "Hardware Virtualization Layer for Ubiquitious Reconfigurable Computing" at the NFS center for HPRC. Finally, it is worth mentioning the EU FP-7 funded project "REFLECT: Rendering FPGAs to Multicore Embedded Computing" [21] which explores the use of Aspect Oriented Programming as an alternative way to deal with high level synthesis for FPGAs.

III. *Cluster Architecture*

In this section, we describe our proposal in each reconfigurable computing domain. We start with architecture base description, after that we explain the programming models support, and finally we present the clusters services for the users.

All domains are interconnected each other. Since design decisions in one domain can affect decisions in the other, we need inter-domain interfaces that limit their responsibilities. In figure 2, we briefly denote these responsibilities, and we will explain each one in next sections.

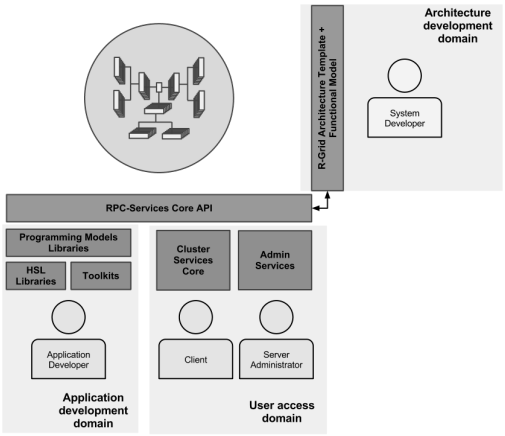

Fig. 2. Interfaces

A. Architecture development domain - Architecture

The physical architecture must offer the maximum flexibility to build reconfigurable computing systems with different topologies, network connections and different computational devices. To obtain that, we offer an *Cluster Architecture* template with construction system rules, and a functional model to fix its behavior.

The *Cluster Architecture* template allows the designer to build a scalable FPGA-based cluster, while functional model defines a several useful features such as: user application repository, automatic application deployment, transparent location and communication, and auto-discover resources, and programming libraries among others.

The template is based on a very simple two role scheme, represented in the figure 3. One of them is centered in the management of platform resources and in the interaction with

the *user access domain*, and the other one is centered in offering computational resources.

The model has another element, the network interconnection, that allows cluster node to communicate between them.

Our *Cluster Architecture* template defines a single instance of the management node, while the number of instances of the computation node can reach high values.

A computational device becomes part of the architecture when it implements a set of services such as communication, announcement, deployment and location. These services, that we define as *computational node kernel services*, represent the lowest access level allowed in the device, for the management node or for the applications.

An application can use this basic services layer. It has simple primitives for send and receive messages inter-task, locate a remote task, local memory resources access, implicit task deployment, and stop/activation task mechanisms. However, the developer can use more sophisticated parallel programming model with layers implemented over these basic services.

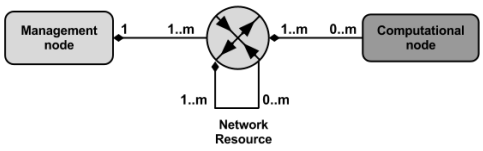

Fig. 3. *Cluster Architecture* template

In figure 4 we can see an example of the cluster physical topology using this template.

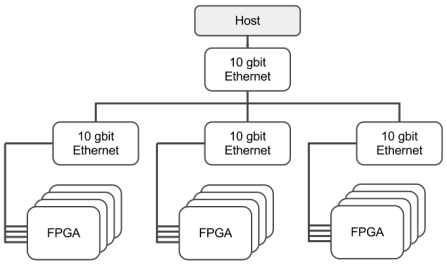

Fig. 4. Architecture Example

The features of each role of the architecture model are summarized below:

- Management role:
  - Applications repository for each user
  - Management of all resources (Autodiscover protocol)
  - Application deployment
  - Transparent remote access
- Computation role:
  - Load partial bitstream
  - Stop and extract/set state of tasks
  - Location of a deployed task
  - Announcement message when start-up
  - Local memory management

Next we described each node of the architectural template.

1) Management node RPC API: User access domain of the *Cluster Architecture* are the main client of the management node services. The management node must offer four services: application repository service, application deployment service, invocation service and auto-discovery service. The application repository and application deployment services are accessed by this methods:

```
—Application Repository Service
void   addApp     (App app);
App[]  getApp     ();
App    getApp     (string name);
void   removeApp  (string name);
void   addBinary  (string appName,
                   string taskName,
                   string version,
                   string model,
                   byte[] bytes);
byte[] getBinaryList (string appName);
void   removeBinary (string appName,
                     string taskName,
                     string version,
                     string model);
—Application Deployment Service
bool startTask (string appName,
                string taskName);
bool stopTask  (string appName,
                string taskName);
```

The application repository services has CRUD methodology (CreateReadUpdateDelete). *addApp, getApp and removeApp* allows to register, read an remove an user application description, and *addBinary, getBinaryList and removeBinary* allows to register, read and remove user applications binaries.

The deployment service has two methods, *startTask and stopTask*, that deploy or stop task application in the system.

The invocation service works such a router, it waits for a externals call over task, and it redirect the call to the final task location.

Auto-discovery service needs a simple protocol. This protocol require two actors: the computational node that sends periodic discovery messages, and the management node that receives broadcast discovery message. This protocol allows discover new computational nodes or detect network or node failures.

2) Computational node RPC API: A computational node must offer five services: partial reconfiguration service, task location service, transparent messages service, auto-announcement service and local memory service. These services create an abstraction layer that allows for the management of different FPGA models in the same way, or any other accelerator type like GPUs. The location and reconfiguration services are accessed as methods:

```
--Application Deployment Service--
int     deploy   (byte[] binary);
int     runTask  (int area);
int     stopTask (int area);


byte[] getState (int area);
int     setState (int area,
                  byte[] state)
void    clean    (int area);


--Application Location Service--
Addr locate         (string name);
void registerTask   (string name,
                     Addr addr  );
void unregisterTask (string name);
```

The *Application Deployment* service method allows loading a binary in a free resource. For this, a *deploy* method is used, this method return the resource identification for location issues. *runTask* send to the task a signal to indicate that it is ready to start its functionality.

In FPGA environment, *deploy* method isolates dynamic reconfiguration from resource location providing transparent dynamic relocation mechanism, that modifies the bitstream to place it in free resources independently of their locations.

To stop a task *stopTask* halt functionality, and *cleanTask* unlock the resource for next use.

getState and *setState* allow state task persistence, only when the task is in halted state.

The transparent messages, auto-announcement and local memory services are accessed by defined signal interface specified in a non-blocking packet based protocol.

In table I we can observe the summary of resources required for the kernel service implementation in a Virtex 5 FX110T FPGA.

Component	Slices	Slices FFs	LUTs	IOs
Location Service	281	159	528	104
Deployment Service	190	167	359	211

TABLE I
RESOURCES

B. Application development domain - Programming models

To exploit the parallelism, the proposed architecture is based on the SPMD (single process, multiple data) parallel programming technique. In the SPMD way, application is split into tasks. These tasks are replicated (if needed) and running at the same time in multiple processing nodes.

From the developers point of view, the first step is to choose a programming model. The concepts and artifacts that offers the programming model allows the user to design the application in accordance with it. Once model has been chosen, the application has been split into tasks in the second step. The intercommunication between tasks is also determined by the programming model. For example, a Remote Procedure Call (RPC) programming model uses method call for the communication process, and the task must expose its functionality by method interface.

Our approach support different programming models through a toolkit, and specific library layers (fig. 2). These layers are built over the same interface and allow to integrate programming models with heterogeneous FPGA devices. To achieve this, each FPGA is managed by a common kernel deployed in a static area as mentioned previously.

The toolkit allows the creation of adapters and stubs of the task interface according to the programming model artifacts. For example, a RPC programming model generates the proxy and skeletons artifacts to provide a method invocation semantic. As it currently stands, the toolkit only support RPC programming model. In next version, we plan to offer MPI and MapReduce programming models. In figure 5 we can observe how the application layer is over different programing models and service platforms.

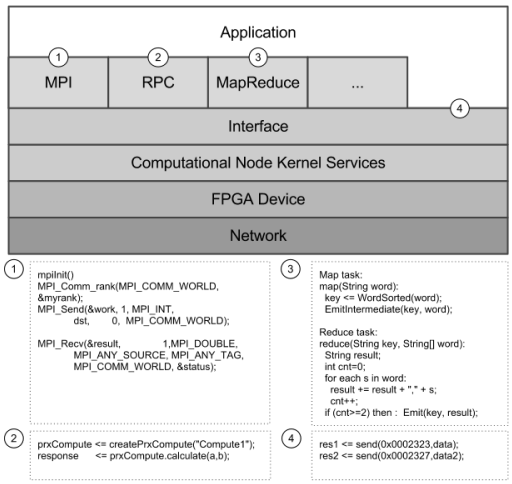

Fig. 5. Programming model

Another advantage of the programing model flexibility is that it allows to use of more than one programming model in the same applications and in the same task. In this way, the developer can choose, task by task, the more adequate programming model.

Development flow

The application developers have a lot of freedom in modeling their application architecture. For example, in our *Cluster Architecture* hardware tasks can intercommunicate each other without the action of a host; this avoids host-coprocessor architecture bottleneck presents in tightly coupled HRPC such CHRECs Novo-G and EPCCs Maxwell systems.

In figure 6 we can see briefly the application development workflow proposed. It has five phases from the applications analysis to the execution. During *Phase 1* the developer will choose the programming model that better suited to his application domain. In *Phase 2* according to selected programming model the application will be modeled as a set of related task graph. This graph shows the relations,

the dependencies and the parallelism between tasks. During *Phase 3* the implementation is performed: each task is described using *High Level Languages*, the corresponding communication adapters are automatically generated through programming model dependent toolkit, and the described task plus the communication adapters are synthesized. Finally, the computational node model is selected and the corresponding *binary files* are generated. In the *Phase 4* the developed application is registered into the cluster repository. Finally, during *Phase 5* the application is deployed from the repository and are executed.

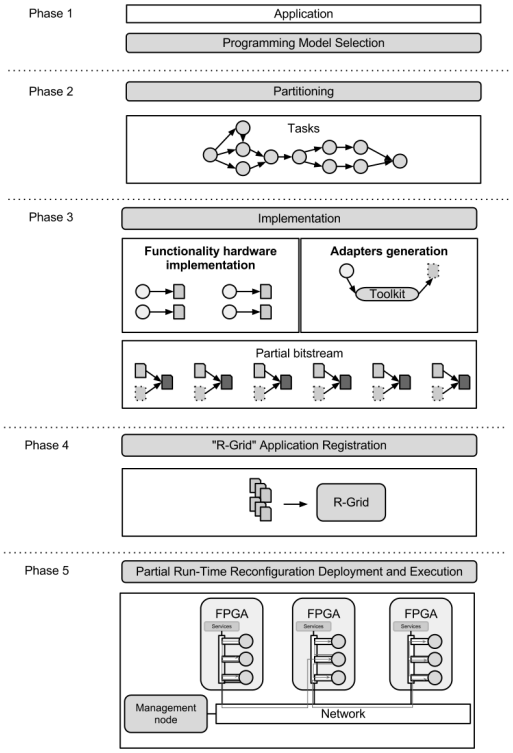

Fig. 6. Application development flow

Figure 7 exemplifies the hardware mapping of an application architecture from the task model. In this example the application architecture is formed by nine tasks based on five roles: two for data partitioning and result storage, and three to pipeline computation. The nine task were deployed in three FPGAs.

C. User access domain - Services

All the facilities described in precedent paragraphs wont be useful if the system does not provided a simple way to end user to exploit all platform resources. For this reason, in the *user access domain* it is necessary that an external access point for clients and system administrators is provided, that include a graphical client application. This application provides client access to system services.

The main functionalities in the user access domain are the processing of external connections, user identification and au-

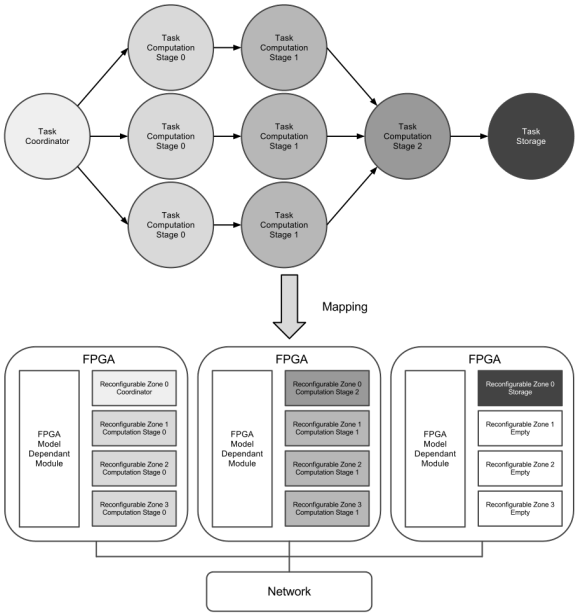

Fig. 7. Application architecture mapping example

thentication, application registry and deployment, accounting, and security. These functionalities are obtained from services provided by the management node as detailed in section III-A1.

In this domain the external services offered to the clients:

- Clients
  - Application repository service
  - Application deployment service
  - Programmatically external access to internal deployed applications
- Administrator
  - System status services
  - System admin services
  - Errors awareness services

IV. EXAMPLES

Grid Services have demonstrated over the years that can be helpful in high-performance computing multi-node exploitation. From end-users point of view, our proposal offers the same services provided in traditional cluster and grid system plus the benefits obtained from the incorporation of reconfigurable computing resources. Nonetheless, the facility of cluster use perceived by user will depend on the quality of final implementation of the described services.

A matrix multiply application was chosen to be accelerated using our approach.

The experiment consists in the implementation of a matrix multiplier in our *Cluster Architecture* platform to analyze the viability of the proposal, evaluating if the models are correctly defined and the benefits obtained in the complete application deployment process.

The programming model selected is RPC. With this model we split the applications in three tasks: partitioning, computation and storage. Figure 8 shows the architecture with the compute task replicated five times. To implement this example we chose two Virtex 5 VLX110T with 512 MB of RAM, Rocket-IO and Ethernet 1 Gbps.

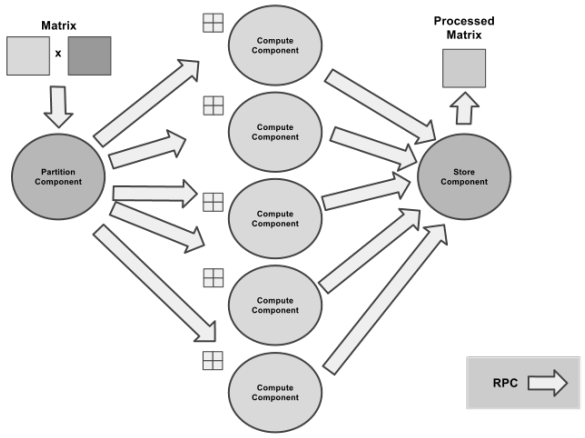

Fig. 8. Logical matrix multiplication architecture

Figure 9 shows time table for different solutions. The hardware has four different configurations: 1x1, 2x2, 3x3 and 4x4. Each one represents a fine-grained parallelism implemented with matrix multiplication array. The solution selected is the 4x4 with five replicated task.

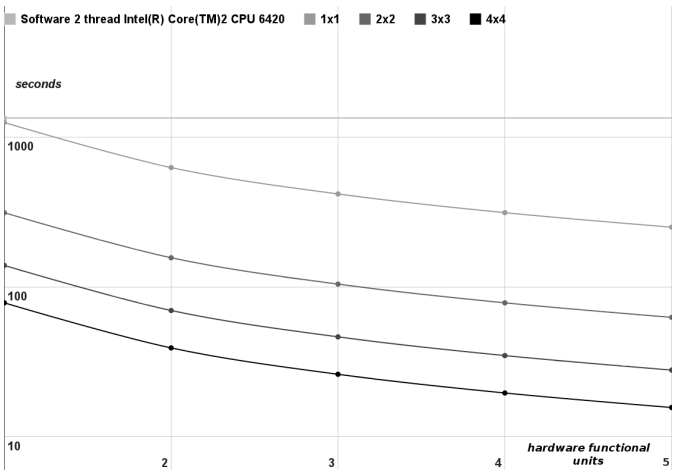

Fig. 9. Matrix multiplication time

The computation time to solve a 5000x5000 matrix multiplication using this reconfigurable infrastructure with five 4x4 computational kernels was reduced from 1340 seconds using software (dualcore Intel Core2 6420 2.13GHz) until 16 seconds, which means 83.75 times faster.

V. Conclusion

Our approach offers an architectural model that allows to exploit heterogeneous cluster of FPGAs. The solution supports the cluster building process, its management, and it simplifies application development and execution.

This complete workflow simplifies the incorporation and exploitation of distributed hardware resources for high-performance computing environment, offering flexible and integrated platform domains to facilitate platform use.

The future work focus in the programming model's library and toolkit support, the development of service kernels for different FPGA models, and a repository of general-purpose tasks.

References

[1] R. Baxter, S. Booth, M. Bull, G. Cawood, K. D?Mellow, X. Guo, M. Parsons, J. Perry, A. Simpson, and A. Trew, "High-performance reconfigurable computing - the view from edinburgh," *Adaptive Hardware and Systems, NASA/ESA Conference on*, vol. 0, pp. 273–279, 2007.

[2] T. Callahan, J. Hauser, and J. Wawrzynek, "The garp architecture and c compiler," *Computer*, vol. 33, no. 4, pp. 62 –69, apr 2000.

[3] S. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. Taylor, and R. Laufer, "Piperench: a coprocessor for streaming multimedia acceleration," in *Computer Architecture, 1999. Proceedings of the 26th International Symposium on*, 1999, pp. 28 –39.

[4] B. Mei, S. Vernalde, D. Verkest, and R. Lauwereins, "Design methodology for a tightly coupled vliw/reconfigurable matrix architecture: a case study," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 2, feb. 2004, pp. 1224 – 1229 Vol.2.

[5] M. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The raw microprocessor: a computational fabric for software circuits and general-purpose programs," *Micro, IEEE*, vol. 22, no. 2, pp. 25 – 35, mar/apr 2002.

[6] C. Ebeling, C. Fisher, G. Xing, M. Shen, and H. Liu, "Implementing an ofdm receiver on the rapid reconfigurable architecture," *Computers, IEEE Transactions on*, vol. 53, no. 11, pp. 1436 – 1448, nov. 2004.

[7] S. G. I. Corp., http://www.sgi.com/products/servers/altix/, 2012.

[8] N. an IBM Company, http://www.netezza.com/data-warehouse-appliance-products/index.aspx, 2012.

[9] C. C. Corporation, http://www.conveycomputer.com, 2012.

[10] C. Inc, http://www.cray.com/products/Legacy.aspx, 2012.

[11] S. M. P. Computing, http://www.sciengines.com, 2012.

[12] X. Meng and V. Chaudhary, "A high-performance heterogeneous computing platform for biological sequence analysis," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 9, pp. 1267 –1280, sept. 2010.

[13] Y.-T. Hwang, C.-C. Lin, and R.-T. Hung, "Lossless hyperspectral image compression system-based on hw/sw codesign," *Embedded Systems Letters, IEEE*, vol. 3, no. 1, pp. 20 –23, march 2011.

[14] C.-H. Huang and P.-A. Hsiung, "Hardware resource virtualization for dynamically partially reconfigurable systems," *Embedded Systems Letters, IEEE*, vol. 1, no. 1, pp. 19 –23, may 2009.

[15] I. Impulse Accelerated Technology, *Impulse Tutorial: Generating HDL from C Language*, 2009.

[16] I. Page, "Constructing hardware-software systems from a single description," *The Journal of VLSI Signal Processing*, vol. 12, pp. 87–107, 1996, 10.1007/BF00936948. [Online]. Available: http://dx.doi.org/10.1007/BF00936948

[17] D. Galloway, "The transmogrifier c hardware description language and compiler for fpgas," in *FPGAs for Custom Computing Machines, 1995. Proceedings. IEEE Symposium on*, apr 1995, pp. 136 –144.

[18] A. Takach, "Catapult c synthesis: Creating parallel hardware from c++," in *Int. Symp. Field-Programmable Gate Arrays Workshop, 2008. Proceedings*, feb 2008.

[19] I. Berkeley Design Technology, "An independent evaluation of: High-level synthesis tools for xilinx fpgas," BDTI, Tech. Rep., 2010. [Online]. Available: http://www.bdti.com

[20] Altera, http://www.altera.com/b/opencl.html, 2012.

[21] R. Project, http://www.reflect-project.eu/, 2012.