Highly-Scalable Reconfigurable Computing

Roger D. Chamberlain*, Steven Miller†, Jason White*, and Dan Gall†
*Exegy Inc., roger@exegy.com, jwhite@exegy.com
†Silicon Graphics, Inc., scm@sgi.com, dgall@sgi.com

Abstract

Reconfigurable computing, typically deployed using Field-Programmable Gate Arrays (FPGAs), is a technology that is ideally suited to exploit the fine-grained parallelism that exists in many algorithms and applications. SGI is a manufacturer of large-scale, high-performance computer systems and has recently developed an FPGA-based node that is integrated into its interconnect architecture. Exegy has deployed several codesigned applications onto commodity hardware (processors and FPGAs). Here, we describe our experience deploying Exegy applications on SGI systems, including both the application development process as well as performance results.

1. Introduction

Commercially available FPGAs have recently achieved capacities that enable the deployment of production applications within an individual chip. The reconfigurable logic within an FPGA makes it ideal for exploiting the fine-grained parallelism within an algorithm. For example, if a computation can concurrently execute 100 accumulation operations, an FPGA implementation of that computation can deploy 100 adders and perform all 100 accumulation operations in parallel, all in a single chip. On the other hand, systems that include large collections of FPGAs have previously either been dedicated to specialized purposes or non-existent.

SGI's ccNUMA (cache-coherent non-uniform memory architecture) [1] global shared memory system architecture is the basis for its general-purpose Altix® HPC systems, which can scale up to thousands of compute nodes. The NUMAlink4 is the current generation physical interconnect that ties these systems together. SGI has recently developed an FPGA-based node for the NUMAlink4 interconnect that supports a 6.4 GB/s data rate into and out of the FPGA [2].

At Exegy Inc. and Washington University in St. Louis, we have been quite active in developing applications that are deployed on FPGA platforms (i.e., codesigned for hardware/software deployment). The application focus has been on computations that involve very large (multiple terabyte) unstructured data sets [3]. On a single node, performance gains for complex text search applications exceed two orders of magnitude over software-only solutions [4]. Other codesigned applications include encryption, signature hashing, data mining, biosequence similarity search [5], etc. The scalability of the SGI system enables further significant performance gains on these applications.

In this paper, we describe the process of porting the Exegy application set to the SGI system. We first describe the SGI system architecture. This is followed by an exposition of an example set of applications. We then describe the application development process, both as it relates to the original application development and the porting of those applications to the new environment. We finish with a discussion of the performance implications of this work; including measured results for individual nodes as well as a discussion of scaling up to large node counts.

2. SGI System Architecture

Large multi-processor systems are very difficult to integrate tightly. The most common approach in use today is to acknowledge this directly and use a traditional network of smaller computers in a cluster environment. This approach is flawed in many ways. Memory is segmented, limiting the size of datasets and requiring multiple copies of data exist in still-expensive RAM. (Yes, memory is drastically less expensive than in years past, but dataset growth has outpaced that decline.) Further, data must be actively passed by one processor to another, taking time away from workload computations and requiring more context switches and kernel calls. Of course, such groups of machines must also keep track of which node is using which data.

Silicon Graphics' system architecture is centered about a cache-coherent non-uniform memory architecture (ccNUMA). In SGI's Altix line of Linux supercomputers, ccNUMA works with and augments features in Intel Itanium processors made for small, symmetrical multi-processor systems. In essence, none of the processors in an SGI supercomputer have any memory. The SGI SHUB is the single-chip ccNUMA controller that interconnects processors to memory and the NUMAlink interconnect. It owns all of the memory in a NUMA node, and arbitrates access to it from every processor in the system. (This is illustrated in the left half of Figure 2, which shows a 2 processor Altix 350.) The NUMAlink4 interconnect allows the SHUBs to cooperate, growing the system organically. The result from the user's perspective is a single pool of memory on a single large single supercomputer with maximum memory access latency around 900 ns across up to 32 TB of system RAM. Access time is variable because of the nature of the NUMA interconnect. This is part of the meaning of non-uniform memory architecture. Gone are all of the negatives associated with traditional clusters.

The programmable logic product, named "Reconfigurable Application Specific Processor" or RASC is a single large Xilinx FPGA and associated QDR SRAM buffers (illustrated in the right half of Figure 2). SGI has also built a good device manager and driver, along with netlists for a "Core Services" algorithm wrapper that abstracts the hardware from the algorithm developer. The ability to single-step the algorithm is provided, as is FPGA integration with the GNU debugger. Further, there is a good co-simulation environment that works under both VCS and ModelSim. All of this provides a fully usable, mature product with a similar programming experience to existing products but with compelling advantages in bandwidth and a clear path to more robust programming models.

SGI's PCI-X IO, graphics and programmable logic "peripherals" are also integrated directly into the NUMA fabric, making the term "peripheral" inaccurate [1]. In fact, they are as tightly coupled as the processors, being directly connected via a NUMA controller ("TIO") into the NUMAlink fabric with a pair of 6.4GB/s NUMAlink4 channels.

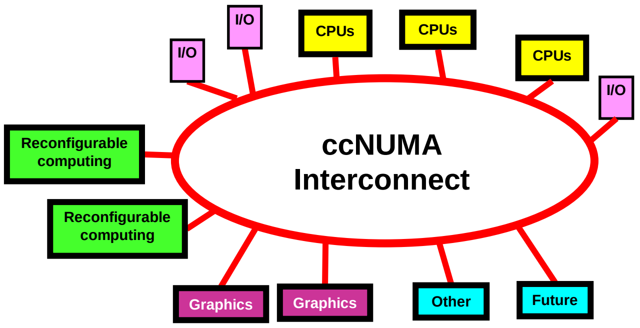


Figure 1. SGI Altix system architecture.

Figure 1 shows a high-level view of the Altix ccNUMA architecture. The CPU is no longer central, but is a resource like any other. These systems scale into the thousands of compute nodes.

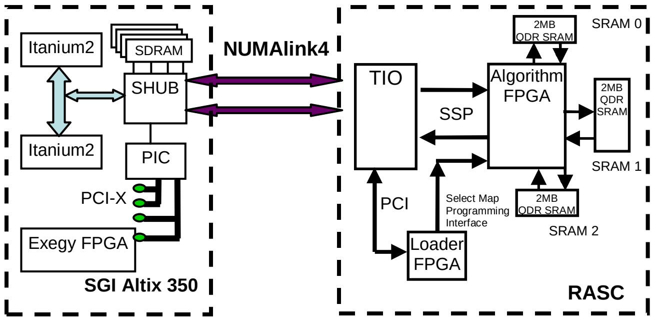


Figure 2. Prototype configuration.

Figure 2 shows the configuration of our experimental prototype system. A 2 processor Altix 350 processor node is connected to a prototype RASC node via the NUMAlink4 interconnect. The prototype configuration is very simple, but can be almost arbitrarily complex as the NUMAlink4 is a general purpose fabric. The dashed block on the left is the processor node, and on the right is the prototype RASC.

In addition to the reconfigurable computing resources made available via the RASC, we are also investigating the use of reconfigurable logic that is attached via the PCI-X bus. This is illustrated in the figure as an Exegy FPGA on the PCI-X bus of the Altix node. This configuration can scale as well, since SGI provides nodes for the NUMAlink4 interconnect that have several PCI-X busses per node.

3. Exegy Applications

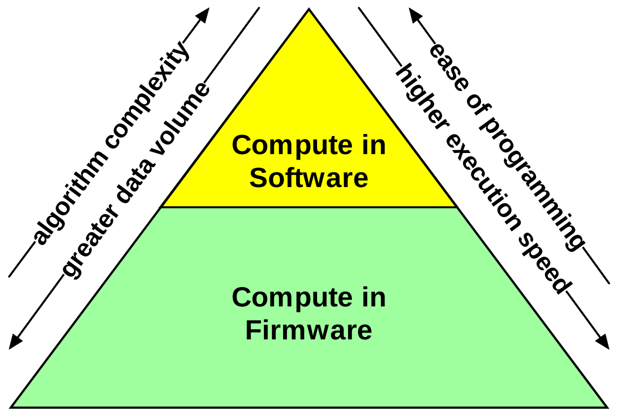
Algorithms that can benefit from reconfigurable hardware generally have one or more of the following properties:

- high levels of fine-grained parallelism,
- relatively small quantities of temporary state,
- simple, regular algorithmic structure.

As a general rule, applications will often have a portion of their constituent algorithms with the above properties and a portion without. The challenge associated with deploying applications on reconfigurable hardware is to perform appropriate application decomposition, essentially solving a hardware/software codesign problem.

The principles that guide the codesign exercise are illustrated in Figure 3. Those portions of the application that require higher performance and deal with large data volumes are serious candidates for deployment in reconfigurable hardware (indicated as firmware in the figure). Those portions of the application that have greater algorithmic complexity and can therefore benefit from the simpler design environment available are targeted for deployment in software. As in any design exercise, there is considerable judgment involved on the part of the designer just where the boundaries lie.

Reconfigurable computing provides performance benefits when the resulting design generally reflects the impression presented by the figure. When the bulk of the application (measured either by data volume and/or original compute time required) is migrated to firmware, the performance gains are maximized. If the converse is true (little of the application moved to firmware), the effort required to perform the design and deployment are often not warranted.



Application Deployment Pyramid

Figure 3. Application decomposition.

We have performed this codesign exercise on a number of example applications. These include:

- exact text search
- approximate text search [4]
- biosequence search (i.e., BLAST) [5]
- structured record search
- encryption/decryption (i.e., 3DES [6])
- signature hashing (i.e., MD5 [7])
- science data mining
- compression/decompression

All of the above applications are either already deployed or under development.

We illustrate application decomposition using the example of an approximate search application. Consider a search (within a set of unstructured text files) for matches on the following query:

```
((Cardinals NEAR[200] Baseball) AND
     (DC United NEAR[200] Soccer))
```

This query expresses the following conditions: (1) the string “Cardinals” is found within 200 characters of the string “Baseball”; (2) the string “DC United” is found within 200 characters of the string “Soccer”; and (3) both conditions (1) AND

(2) hold. This example query is illustrated in tree form in Figure 4.

The current set of combining operators supported include AND, OR, NOT, NEAR, and ANDTHEN. The operators AND, OR, and NOT perform their traditional Boolean logic functions at the file level. The operator NEAR is equivalent to AND with the additional constraint that the matching strings (or subexpressions) must be within a given distance to one another in the file. The operator ANDTHEN is equivalent to NEAR with the additional constraint that the first term must be earlier in the file than the second term (i.e., *x* ANDTHEN *y* imposes a precedence constraint on *x* and *y*). For example, the query:

```
((Cardinals ANDTHEN[200] Baseball) AND
        (DC United ANDTHEN[200] Soccer))
```

requires that "Cardinals" precede "Baseball" and that "DC United" precede "Soccer," retaining the 200 character distance constraint.

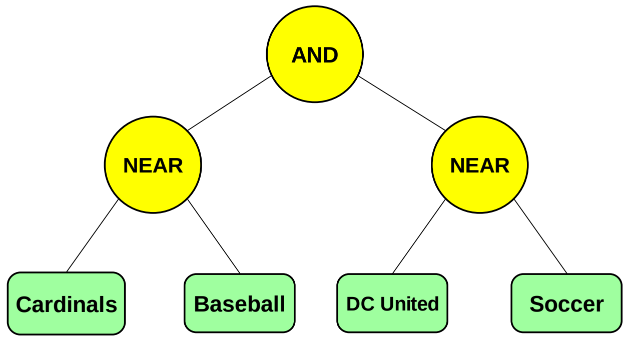


Figure 4. Text search application example.

Our system supports approximate matches of three forms. First, character matches can be case insensitive; second, individual query characters can be designated as wildcards in which case any character will match; and third, a count, *k*, of allowed mismatched characters can be provided in which case up to *k* characters in the string can be wrong and the string still considered to be a match.

In our implementation of this approximate search application, the individual keywords in the query (i.e., "Cardinals," "Baseball," "DC United," and "Soccer") are found through firmware (including any allowed approximate matches) and the logical combinations of these results (i.e., the AND and NEAR operators) are computed in software. In this way the firmware considers the bulk of the raw data, and the software operates on results returned by the firmware.

4. Application Development Environment

While the previous section deals with the high-level issues of what functionality is executed in software vs. firmware, the actual deployment of an application requires that all the low-level issues be addressed as well. Software must be developed and its correctness verified. Firmware must also be developed and tested. A crucial piece of the puzzle is how the software and the firmware interact with each other, i.e., how data and control information is communicated between the two computational environments.

The application development work for all of the applications listed in the previous section was originally targeted at the Exegy K•Appliance™, a network appliance that incorporates both magnetic storage and reconfigurable logic. The application development framework for the K•Appliance is illustrated in Figure 5. The outer two layers of the framework represent the application-specific components of an implementation, the software on the top layer (deployed on a general-purpose processor) and the firmware on the bottom layer (deployed in reconfigurable logic). The next layer up from the bottom is also deployed in the reconfigurable logic and represents the service functions needed to implement the hardware/software interface. Above this level are two software layers that represent the platform-specific software components, both within the OS and in user space.

The intent of this development framework is to enable codesigned applications (i.e., the top and bottom layers) to be easily ported to alternative platforms without modification. Any platform-specific portions are constrained to the middle three layers of the framework.

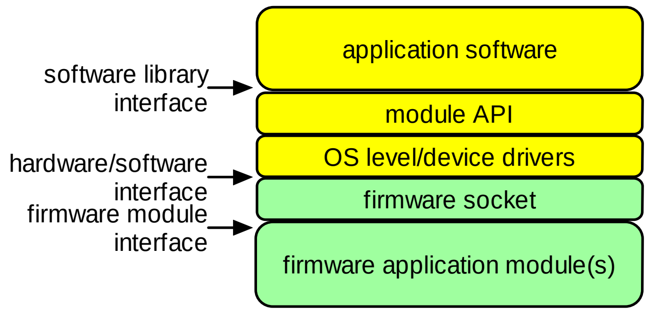

application software
module API
OS level/device drivers
firmware socket
firmware application module(s)
software library interface
hardware/software interface
firmware module interface

Figure 5. Application development framework.

We are targeting two architectural approaches to deploying codesigned applications on the SGI platform, resulting in a pair of platform configurations. The first approach is insertion of a board that contains a Xilinx FPGA into a PCI-X bus slot on the Altix. This first platform configuration is closest in practice to the internal architecture of the Exegy K•Appliance, and is currently operational. Porting of the middle three layers of the development framework was relatively straightforward, and the applications themselves (top and bottom layers) did not need to be altered at all. The performance results presented in the next section are from this platform configuration. The second approach is the use of the RASC directly attached to the NUMAlink4 system interconnect. This second platform configuration is no longer limited by the use of the PCI-X bus for data delivery to and from the FPGA. We are currently in the process of porting the middle three layers of the development framework to this platform configuration, and we anticipate no need to alter the applications themselves. The result is that applications that are developed within the context of the development framework of Figure 5 can execute directly on both platform configurations.

5. Performance Results

In this section we present measured performance results from two specific applications among those listed in Section 3. The first is the approximate search application, in which a large text corpus is searched for a set of keywords. The data includes Shakespeare's collected works, a variety of song lyrics, the publicly available Enron emails, the full Red Hat ES 4.0 distribution (unpacked and in ISO), a body of commercial fixed-length record data, and several other miscellaneous items.

With the firmware portions of the text search application deployed in the FPGA on an Exegy card in the PCI-X bus of the Altix, we achieved a data throughput of 815 MB/s. This throughput compares quite well to the theoretical maximum throughput possible with this configuration (the 1 GB/s supported by the PCI-X bus). We anticipate increased throughput once this application is executing on the RASC.

Figure 6 compares the execution times for a single processor to a single FPGA on an encryption application (3DES encryption [6]). The software implementation is the widely utilized *libdes* library developed by Young [8]. As one would anticipate, the completion time for both implementations is linear in the size of the data to be encrypted. The distinction, however, is the slope of the line. For this application, the FPGA outperforms an individual processor by greater than a factor of 40.

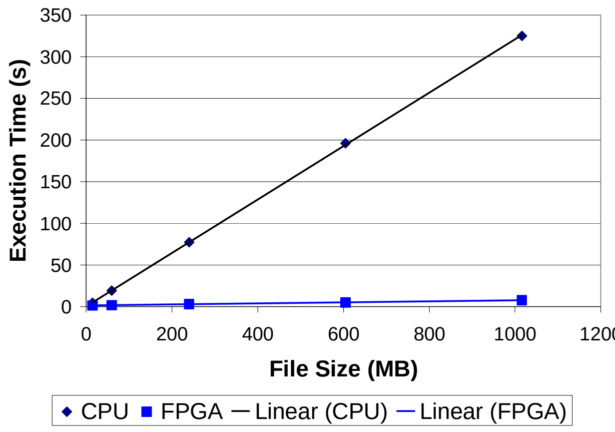


Figure 6. Encryption performance.

In general, the data throughput supported by an individual FPGA on an Exegy card is limited by the PCI-X bus speed. The RASC supports a higher data rate into and out of the FPGA, which will improve the performance of I/O bound applications (e.g., text search). In addition, the RASC communications capability is full duplex, enabling concurrent data flow both into and out of the reconfigurable logic. This is important for applications such as encryption, where the outbound data volume is equal to the inbound data volume.

While the above performance figures are for a single FPGA, the true power of the system is realized as the number of FPGA nodes is scaled up. The SGI Altix line can support up to thousands of nodes, and with one to two orders of magnitude performance gain at the per node level, applications can potentially achieve 5 orders of magnitude performance gain over single processor software implementations through the use of scalable reconfigurable computing. The SGI Altix platform directly supports this level of performance.

Of course, an important consideration when scaling up a system is the parallelism available in the applications themselves. We have two factors working in our favor here. First, the SGI NUMAlink4 interconnect supports a very high data rate (6.4 GB/s per endpoint) with low latency. This interconnect performance is among the best available, anywhere. Second, many of our applications of interest are inherently scalable via data partitioning. Subsets of the data are assigned to distinct processing resources and the results aggregation is a very small portion of the overall task (successfully avoiding the gotchas associated with Amdahl's Law).

6. Conclusions and Future Work

We have presented a system that supports reconfigurable computing at unprecedented scales, providing for thousands of nodes. We have also ported a number of real, production applications to this system, presenting performance results for a pair of these applications.

One of the strengths of this work is the ability for codesigned applications (those that use both traditional software as well as reconfigurable logic to execute the application) to be ported from one platform to another without alteration. All of the applications described here were originally developed on the Exegy K•Appliance under a standard application development framework. By porting only the interior layers of the development framework to a new platform, the applications themselves will run on both platforms unaltered.

The framework porting task is complete for one of the two system configurations described and is in progress for the other configuration. Once this port is ready, we will quantify application performance on a scaled system, assessing the overall performance levels achievable on a highly-scalable, reconfigurable computing system.

[1] http://www.sgi.com/servers/altix/

[2] Steven Miller, "Requirements for Scalable Application Specific Processing in Commercial HPEC," in *Proc. of High Performance Embedded Computing Workshop*, September 2004.

[3] Roger D. Chamberlain and Ron K. Cytron, "Novel Techniques for Processing Unstructured Data Sets," in *Proc. of IEEE Aerospace Conference*, March 2005.

[4] Mark Franklin, Roger Chamberlain, Michael Henrichs, Berkley Shands, and Jason White, "An Architecture for Fast Processing of Large Unstructured Data Sets," in *Proc. of IEEE 22nd Int'l Conf. on Computer Design*, October 2004, pp. 280-287.

[5] Praveen Krishnamurthy, Jeremy Buhler, Roger Chamberlain, Mark Franklin, Kwame Gyang, and Joseph Lancaster, "Biosequence Similarity Search on the Mercury System," in *Proc. of IEEE 15th Int'l Conf. on Application-Specific Systems, Architectures and Processors*, September 2004, pp. 365-375.

[6] ANSI, "Triple Data Encryption Algorithm Modes of Operation," American National Standards Institute X9.52-1998, July 1998.

[7] Ronald L. Rivest, *The MD5 Message Digest Algorithm*, Internet RFC 1321, April 1992.

[8] Eric Young, *libdes*, http://www.openssl.org