2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), and 2015 IEEE 12th International Conf on Embedded Software and Systems (ICESS)

XGRID: A Scalable Many-Core Embedded Processor

Volkan Gunes and Tony Givargis
Center for Embedded Computer Systems
University of California, Irvine, USA
{vgunes, givargis}@uci.edu

Abstract—**The demand for compute cycles needed by embedded systems is rapidly increasing. In this paper, we introduce the XGRID embedded many-core system-on-chip architecture. XGRID makes use of a novel, FPGA-like, programmable interconnect infrastructure, offering scalability and deterministic communication using hardware supported message passing among cores. Our experiments with XGRID are very encouraging. A number of parallel benchmarks are evaluated on the XGRID processor using the application mapping technique described in this work. We have validated our scalability claim by running our benchmarks on XGRID varying in core count. We have also validated our assertions on XGRID architecture by comparing XGRID against the Graphite many-core architecture and have shown that XGRID outperforms Graphite in performance.**

Keywords—Multi-core, Many-core, Embedded Processors, System-on-Chip Architectures

I. Introduction

Embedded systems have an important place in our daily lives. Compute demands of embedded systems have increased in recent years for many electronic devices, including but not limited to mobile devices, consumer appliances, network devices, and military applications. With the growing popularity of mobile and real time technology, this increase in demand is expected to continue for specialized embedded systems to support a wide range of new applications.

To satisfy increasing demands for compute cycles, a move toward multi-core processing was unavoidable [1]. Multi-core processors make improvements in the performance by increasing the number of the processor cores on a single chip, with each core operating at an ideal clock speeds in order to meet overall power and thermal constraints [2].

We draw a distinction between a multi-core system, one that is limited to 8 or less cores, and a many-core system that can scale to tens, hundreds, and even thousands of cores on a single chip. A key requirement for any many-core architecture is its ability to scale efficiently. With increases in the number of cores on a single chip, the performance of the overall system becomes limited by shared resources such as buses and the memory subsystem [3].

In this paper, we present a scalable many-core processor, intended for embedded applications. Our many-core embedded processor is named XGRID. Further, we outline a mapping strategy to efficiently map applications to XGRID. The contributions of this paper are:

- Introduction of a scalable many-core embedded processor adopting 2D grid network, inspired by a novel FPGA-like interconnect network
- Optimal mapping of benchmark applications onto target XGRID architecture

We describe a comprehensive simulation environment for XGRID. Our simulation platform, in addition to offering a cycle accurate functional execution environment, provides detailed performance results to better guide the application mapping process.

II. Related Work and Motivation

Various authors have investigated architectural issues related to many-core interconnect networks, particularly bus based and NoC based approaches [4], [5], [6], [7], [8], [9]. As outlined in the earlier studies, bus based architectures are not scalable with increasing number of cores. Hence, NoC architectures are proposed as a solution to the limitations of bus based architectures [10]. NoC architectures offer some advantages as well as some challenges. For example, switch units, network interfaces, and inter-switch wires result in substantial silicon area overhead. Increased networking complexity as well as the number of interconnected cores within an NoC introduce a considerable trade-off between area and performance [11].

Due to the limitations of bus interconnect and challenges of NoC interconnect, we propose a novel FPGA-like many-core architecture which combines positive attributes of bus based (i.e. power efficient and deterministic) and NoC based interconnects (i.e. scalable and flexible). The simple and low-clock speed nature of XGRID makes it inherently power efficient.

Field-Programmable Gate Arrays (FPGAs) have suffered increased power consumption as a result of large scale dynamic switching networks [12]. Certainly, static routing has some benefits over dynamic routing. For example, in static routing, the path for data packet transmission between two destinations is always known precisely and can be controlled precisely at compile time. Therefore, we propose the XGRID architecture that has a low-cost static interconnect network, making it compile-time configurable with minimal power overhead.

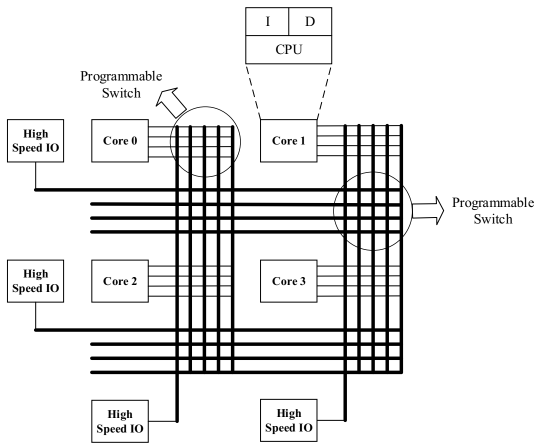

Fig. 1. An instance of XGRID with 2x2 cores,
I: Instruction Memory and D: Data Memory.

III. The Xgrid Architecture

Our embedded many-core processor platform integrates processing cores, on-chip memory per core, FPGA-like interconnection network and serial high-speed I/O units. We call our architecture XGRID, as it consists of a two dimensional grid of homogenous cores. Each XGRID core strictly follows a Reduced Instruction Set Computer (RISC) architecture. Specifically, cores follow a standard five-stage instruction pipeline (fetch, decode, execute, memory-access, and write-back) and lack the branch prediction and out-of-order execution capabilities. Per core flash memory is used to store program instructions, making XGRID in-system programmable. Each core maintains a dedicated instruction and data cache. Each core operates at a relatively low clock frequency, namely, 100 MHz to 500 MHz. As a result, the XGRID cores are lightweight and, hence, low power. The compute performance of XGRID comes from the scalability in terms of the number of cores that are utilized to execute parallel algorithms.

In XGRID, communication between cores is achieved via an FPGA-like interconnection network. FPGAs use rows and columns of buses with programmable switching fabrics at the intersections of the row/column buses to route input/output of logic-blocks. XGRID replaces the FPGA logic-blocks with cores and otherwise adopts the FPGA interconnect fabric for all communication among the cores. As previously mentioned, the difference from an FPGA interconnect is that XGRID uses a static interconnect network that is compile-time configurable.

An instance of the XGRID interconnection network with two rows and two columns is shown in Fig. 1. The figure shows row and column buses in the interconnect network, represented as thick lines. Each core has *N* word-size ports to send or receive data over the buses. The port connections are represented as thin lines in Fig. 1. Communication buses are dedicated, during the programming phase, for bi-directional I/O between two cores with deterministic transmission rates. Appropriate switches need to be set to establish a communication between a pair of cores. This programming, as with FPGA-programming, is performed during the design phase, and the programming bit stream is stored in an on-chip flash memory, making the XGRID communication infrastructure in-system programmable.

XGRID uses a strict message passing system of communication among cores. The cores can communicate with each other via an inter-core communication facility. This facility provides, to the software, a primitive instruction, called XPORT, which is used to send or receive data among cores. Higher-level software routines can be built on top of this instruction to facilitate appropriate transfer capabilities, such as block transfer protocols.

We call the established path between two cores a communication channel. Communication channels include bidirectional buffers to maximize instantaneous throughput among cores. The message sent by a core is of word-size. Since buffer size is limited, a sending core blocks when its send buffer is full. Likewise, a receiving core blocks when its receive buffer is empty. The blocking nature of sending (*XPORT=value*) or receiving (*value=XPORT*) are buffer synchronized, using a consumer-producer message passing scheme.

A major advantage of XGRID is that it avoids global caches and their associated coherency problem as well as shared memory infrastructure having complex on-chip bus and memory controllers. In this sense, XGRID is scalable, as the cost of additional rows or columns scales linearly, namely, consisting of the cost of the new cores and FPGA-like communication fabric.

IV. Application Mapping

We follow a general scheme for our design flow of application mapping. This scheme includes application modeling and the actual mapping stages. Our holistic application mapping is shown in Fig. 2. First of all, a sequential application is manually partitioned and represented as a Kahn Process Network (KPN) where each process corresponds to a partition and each communication channel between two processes corresponds to a connection between two partitions. Then, the hardware architecture properties of XGRID and the benchmark application model represented as a KPN are fed into an ILP generator. The ILP generator produces ILP formulas consisting of variables, constraints, and constraint equations related to XGRID architecture. Then ILP solver generates a solution that reflects an optimal mapping and routing of the application to XGRID. The interconnect template creator takes this ILP solution and evaluates it against any remaining communication channel constraints (i.e. those not directly captured by the ILP) of the XGRID. For example, since each core in XGRID has a limited number of ports, every communication channel in the KPN representation may not be mapped into XGRID. Therefore, the KPN representation may need a modification to fulfill remaining requirements. The interconnect template creator decides whether or not to accept the ILP solution. If the solution is not accepted, the KPN is modified and the process repeats. Here, the ILP

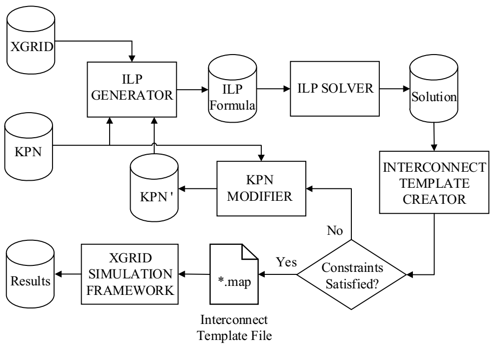

Fig. 2. Holistic view of application mapping onto XGRID.

generator follows the same flow by taking the modified KPN (KPN') as an input for the application model. If the solution is accepted, then interconnect template file is created. Subsequently, the XGRID simulation framework takes care of simulation process to obtain the experiment results. The details of the XGRID simulation framework can be reached in [15].

The KPN captures the communication behavior of the application. We assume applications are implemented using a parallel algorithm, where each task is represented as a process in a KPN. Each node of a KPN corresponds to one process (i.e. a task or partition from the parallel algorithm). The communication between processes is accomplished via channels which include unbounded FIFO queues. In a pure KPN, a sender never blocks, as the size of the FIFO is infinite. However, the receiver may block pending data to be sent by the sender. On the other hand, the sending processes may be blocked in XGRID, hence introducing a performance issue rather than a correctness concern. The KPN is annotated with process compute requirements and channel communication requirements. These annotations are obtained from the application and/or algorithm.

The mapping phase optimally maps each process of a KPN to an XGRID core. Moreover, the mapping phase automatically establishes point-to-point communication channels, according to the KPN, by programming appropriate interconnect switches of the XGRID. The mapping problem is formulated as a set of constraints and an objective function in the form of integer linear equations as described in [15]. The objective function, which we are aiming to minimize, is the overall communication cost of the system configuration. The ILP solver minimizes this total cost function based on the given constraints and equations.

V. EXPERIMENTAL RESULTS

We have selected a number of benchmarks to validate the XGRID architecture, performance profiling, and application mapping algorithms. In particular, we have used 2D DCT (Discrete Cosine Transform), MMUL (Matrix Multiplication), and four different versions of sorting benchmarks. Sorting algorithm benchmarks consist of the QSORT algorithm and three parallel algorithms based on QSORT, namely, PARALLEL-QSORT [13], HYPER-QSORT [13], and PSRS-QSORT [13].

For each benchmark, we have extracted a KPN and used the ILP approach, presented earlier, to obtain a mapping of the algorithm to our XGRID processor. The specific XGRID processor, used in our experiments, is a 4x4 grid of 32-bit cores, each core having eight 32-bit ports. The communication infrastructure is composed of four 32-bit buses spanning the space between any two adjacent rows of cores. Likewise, the communication infrastructure is composed of four 32-bit buses spanning the space between any two adjacent columns of cores. There are a total of four serial input units, and a total of four serial output units for off-chip communication. Each core has a 1 MB data cache and a 64 KB instruction cache.

Fig. 3 shows the performance speedup of 2D DCT relative to a single-core implementation for various input sizes. The performance degradation of 2D DCT is expected in case of larger input matrix sizes since I/O wait has considerable effect on the performance for them. On the average, our 16-core XGRID achieved 9X improvement in the performance of DCT. Fig. 4 shows the performance speedup of MMUL relative to a single-core implementation for various input sizes. On the average, our 16-core XGRID achieved 3X improvement in the performance of MMUL. Moreover, the speedup increased as the input size got larger, because the initial cost of reading the matrices relative to the cost of multiplication diminished.

PSRS-QSORT is the best among our parallel sorting algorithm benchmarks based on QSORT. It does an excellent job balancing the number of elements sorted by each core [13] and I/O wait takes very little time. The results for our parallel sorting algorithm benchmarks can be reached in [15]. Our results point out the importance of efficient parallel programming on many-core architectures. So, the scalability of the different parallel sorting algorithms demonstrates the need for careful algorithm design in many-core implementations.

In order to validate our scalability claim about XGRID, we ran 2D DCT and MMUL benchmarks on XGRID varying in core counts. Fig. 5 and Fig. 6 show the execution time of 2D DCT and MMUL benchmarks on XGRID with different core counts, respectively. 2D DCT benchmarks scale well in all core categories. It is expected since DCT is a computation intense application and the computation dominates the communication in all categories. MMUL benchmarks scale well in all categories except the last one (i.e. 256 cores) where the performance bottleneck occurs. The reason is that MMUL is a communication intense application therefore the communication time dominates the computation time for increasing number of cores and, as a result, causes a decrease in performance.

We also validated our assertions on XGRID architecture by comparing XGRID against Graphite many-core architecture. Fig. 7 and Fig. 8 show the execution time comparison of 2D DCT and MMUL benchmarks, respectively. For the sake of fairness in comparison, we

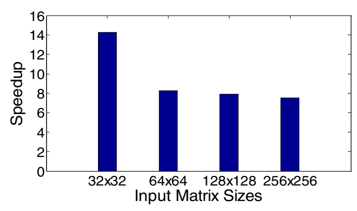

Fig. 3. 2D DCT speedup (single-core vs XGRID)

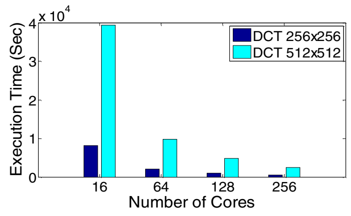

Fig. 5. 2D DCT running on XGRID of various size

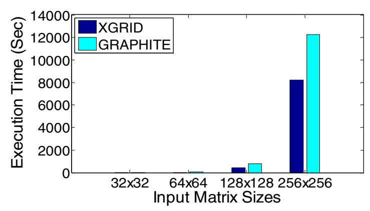

Fig. 7. 2D DCT comparison (XGRID vs GRAPHITE)

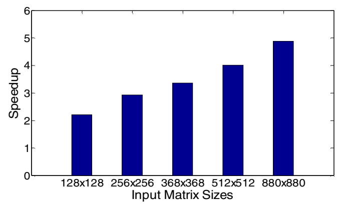

Fig. 4. MMUL speedup (single-core vs XGRID)

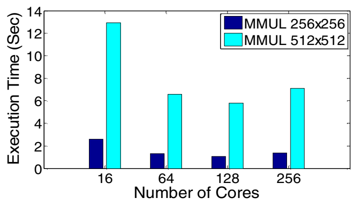

Fig. 6. MMUL running on XGRID of various size

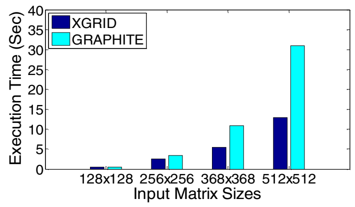

Fig. 8. MMUL comparison (XGRID vs GRAPHITE)

designated same features for both architectures [15]. We ran some of our benchmarks on an open-source simulator for Graphite many-core architecture [14], to justify the performance of our simulator for XGRID. Graphite integrates a set of homogeneous tiles inter-connected by a mesh on-chip network that manages the routing of network packets. Each tile contains a processing core, a memory module, and a network switch [8]. The reasons why we chose the Graphite among existing many-core architectures are that there is an open-source simulator for Graphite and it is well-documented. In addition to that, the Graphite has the tile processor architecture with a mesh on-chip network. The results show that XGRID outperformed Graphite in execution time in all the cases. The details of overall simulation results can be reached in [15].

VI. Conclusions

In this paper, we introduced the XGRID embedded many-core processor that makes use of an FPGA-like interconnection network. The XGRID architecture offers numerous advantages, such as low power consumption (due to cores inherently lightweight), hardware supported message passing, and most importantly, scalability as more processing cores are added. We further describe an application mapping algorithm based on Kahn Process Networks (KPNs) and Integer Linear Programming (ILP) to aid in the mapping of applications on XGRID.

Our experimental results are very encouraging. A number of parallel benchmarks are evaluated on XGRID processor using the mapping technique described in this work. Results show an average of 5X speedup, a maximum of 14X speedup, and a minimum of 2X speedup, across all the benchmarks. We observe that, in addition to the need for a scalable architecture, scalable parallel algorithms are required to exploit the compute power of many-core systems. We have validated our scalability claim by running our benchmarks on XGRID varying in core count. We have also validated our assertions on XGRID architecture by comparing XGRID against the Graphite many-core architecture and have shown that XGRID outperforms Graphite in all performance categories.

Acknowledgement

This work was supported in part by the National Science Foundation under NSF grant number 1016789 and 1136146.

References

[1] Herb Sutter. The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. *Dr. Dobb's Journal*, 30(3), March 2005.

[2] S. Borkar. Thousand Core Chips - A Technology Perspective. In *Proceedings of Design Automation Conference (DAC)*, 2007.

[3] A. Kayi, T. El-Ghazawi, and G. Newby. Performance issues in emerging homogeneous multicore architectures. In *Proc.of Simulation Modeling Practice and Theory*, Vol:17, issue:9, pp.1485-1499, 2009.

[4] W. Zhang, et al. Design of a Hierarchy-Bus Based MPSoC on FPGA. In *Proc. of Inter. Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pp. 1966-1968, 2006.

[5] E. A. Carara, et al. HeMPS - a Framework for NoC-based MPSoC Generation. In *Proc. of IEEE Inter. Symposium on Circuits and Systems (ISCAS)*, pp. 1345 - 1348, 2009.

[6] S.V. Tota, et al. A Case Study for NoC Based Homogeneous MPSoC Architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, March 2009.

[7] S.V. Tota, et al. MEDEA: a hybrid shared-memory / message-passing multiprocessor NoC-based architecture. In *Proceedings of Design, Automation, and Test in Europe (DATE) Conference*, pp. 45-50, 2010.

[8] J. E. Miller, et al. Graphite: A Distributed Parallel Simulator for Multicores. In *IEEE 16th International Symposium on High Performance Computer Architecture (HPCA)*, Jan. 2010.

[9] S. Bell, et al. TILE64-processor: A 64-core SoC with mesh interconnect. In *Proc. of IEEE Int. Solid-State Circuits Conf.*, pp. 88–598, Feb. 2008.

[10] L. Benini and G. DeMicheli. Networks on Chips: A New SoC Paradigm. *IEEE Trans. on Computers*, vol. 35, no. 1, pp. 70-78, 2002.

[11] U. Y. Ogras, J. Hu, and R. Marculescu. Key research problems in NoC design: A holistic perspective. In *Proc. of CODES+ISSS*, pp. 69–74, Sep. 2005.

[12] L.Shang, A.Kaviani, K.Bathala. Dynamic Power Consumption in Virtex-II FPGA Family. In *International Symposium on FPGAs*, pp.157-164, 2002.

[13] M. J. Quinn. Parallel programming in C with MPI and OpenMP. *McGraw-Hill Higher Education*, 2004.

[14] Graphite Simulator Source Code. [Online]. Available: https://github.com/mit-carbon/Graphite/wiki

[15] V.Gunes and T.Givargis. XGRID: A Scalable Many-Core Embedded Processor. Center for Embedded Computer Systems (CECS) at UCI, Technical Report # TR 13-03, 2013. Retrieved from http://cecs.uci.edu/files/2013/04/TR-13-03.pdf