Center for Embedded Computer Systems
University of California, Irvine

XGRID: A Scalable Many-Core Embedded Processor

Volkan Gunes[1,2] and Tony Givargis[1,2]

[1]Center for Embedded Computer Systems

[2]Department of Computer Science

University of California, Irvine

Irvine, CA 92697, USA

{vgunes, givargis}@uci.edu

CECS Technical Report # TR 13-03

April 22, 2013

XGRID: A Scalable Many-Core Embedded Processor

Volkan Gunes[1,2]

[1] Center for Embedded Computer Systems

[2] Department of Computer Science

University of California, Irvine, CA, USA

vgunes@uci.edu

Tony Givargis[1,2]

[1] Center for Embedded Computer Systems

[2] Department of Computer Science

University of California, Irvine, CA, USA

givargis@uci.edu

ABSTRACT

The demand for compute cycles needed by embedded systems is rapidly increasing. Due to the limitations of single-core processors, a move towards multi-core architectures is unavoidable. 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. We have developed a simulation framework for the XGRID architecture, which provides system performance information. 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. Results show an average of 5X speedup, a maximum of 14X speedup, and a minimum of 2X speedup across all benchmarks. 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

1. 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, operating clock frequency of processors was increased accordingly for years. However, it is no longer feasible to continue gaining improvements through single-core devices by increasing clock speed [1]. Providing higher CPU clock speeds to improve performance results in a non-linear increase in power consumption and this increase generates excessive operating temperatures. This situation requires more advanced cooling systems, adding cost, and decreases the reliability of the overall system [2].

The limitations of single-core processors running at high clock rates have forced the computer industry to shift to a new approach for increasing performance of computer systems, namely, a move toward multi-core processing [1]. Multi-core processors make improvements in the performance by increasing the number of the processor cores on a single chip, with each processor operating at an ideal clock speeds in order to meet overall power and thermal constraints [3].

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 [4]. In particular, cache coherency issues come into play as the number of cores increases beyond small number [5].

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.

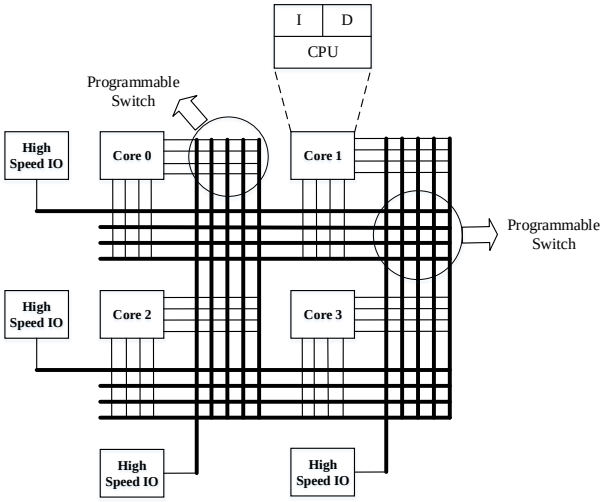


Figure 1: An instance of XGRID with 2x2 cores, I: Instruction Memory and D: Data Memory

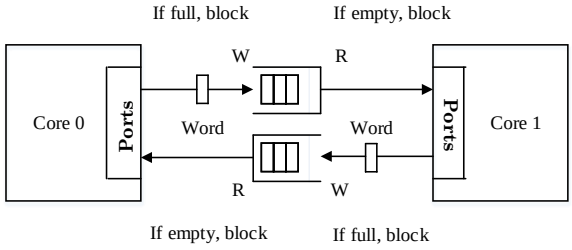


Figure 2: Message Passing Mechanism in a communication channel, W: write and R: read

We describe a comprehensive simulation environment for XGRID. Our simulation platform, in addition to offering a cycle accurate functional execution environment, provides detailed energy and performance results to better guide the application mapping process. The rest of the paper is organized as follows. Related work and motivation is investigated in Section 2. The XGRID architecture is introduced in Section 3. XGRID simulation framework is introduced in Section 4. Application mapping is outlined in Section 5. System energy profiling is introduced in Section 6. Performance analysis and experiments are summarized in Section 7. Conclusions are provided in Section 8.

2. RELATED WORK AND MOTIVATION

Modern system-on-chip (SoC) design methods are increasingly becoming communication-centric rather than computation centric, so it is expected that the interconnect effect will dominate the performance of many-core SoCs [8]. There are two common interconnect architectures, namely bus based and network-on-chip (NoC) based interconnects. Various authors have investigated architectural issues related to bus based and NoC based approaches. We summarize a number of related articles that address the key components of the XGRID project.

A hierarchy-bus based multi-processor system-on-chip (MPSoC) is studied in [9] by prototyping the architecture on a field-programmable gate array (FPGA) and comparing the results of the MPSoC with a single core processor. A framework for NoC-based MPSoC, called HeMPS is presented in [10] with support for static and dynamic task mapping based on a C/SystemC simulation model of processors and memories. The NoCRay graphic accelerator, an implementation of NoC-based homogeneous MPSoC, is studied in [11] by using a complete design methodology that tackles different aspects of hardware architecture, system level modeling, and programming models. Here, the results are compared with a general purpose single core high-speed processor. The Medea framework, an example of NoC based multiprocessor architecture adopting a configurable hybrid shared-memory/message-passing architecture, is presented in [12] with a fast, cycle-accurate SystemC implementation enabling system exploration by varying several parameters such as the number and types of cores, cache size and policy, and NoC features.

The study in [20] represents a heterogeneous multi-core simulator framework, called MC-Sim, which is capable of accurately simulating a variety of processors, memory, NoC configurations, and application specific coprocessors. Gem5 simulation infrastructure, the combination of the M5[28] and Gems[29] simulators, are proposed in [30] as a full system simulation tool, offering a diverse set of CPU models, system execution models, and memory models. HORNET, a configurable, cycle-accurate multi-core simulator is presented in [31]. Being based on an NoC architecture, it also supports a variety of memory hierarchy and interconnect configurations. Graphite, a distributed, parallel many-core simulator is introduced in [32] for design space exploration of future many-core processors. Being based on the TILE [34] processor architecture with a mesh network, it provides detailed performance metrics of application benchmarks and supports McPAT [33], an integrated power, area, and performance measuring framework.

As outlined in earlier studies, bus based architectures are not scalable with increasing number of cores [13]. Hence, NoC structures are proposed as a solution to the limitations of bus based architectures [14]. NoC structures 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 [15][18]. In an NoC, often times, the interconnect infrastructure consumes more power than other system components [16]. Power consumption can be reduced by decreasing the number of interconnected components; however, this adversely affects overall performance [16]. Scaling down the supply voltage to save power consumption can not sufficiently compensate for higher interconnect infrastructure complexity [17] and need for compute cycles dedicated to the message routing algorithms. Furthermore, in NoCs, non-deterministic communication delays do occur due to communication resource contention [19]. So quality of service (QoS) and congestion control mechanisms are needed to overcome that issue [19].

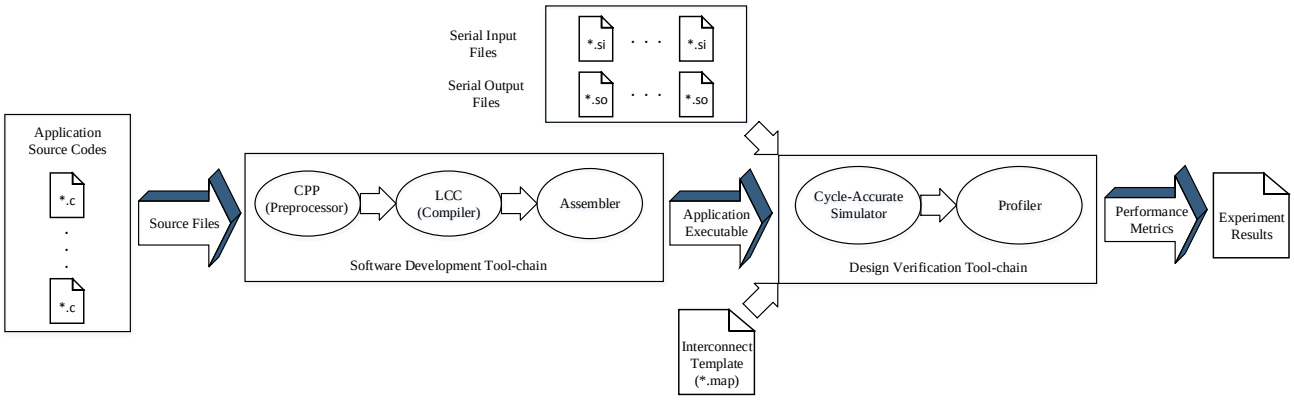


Figure 3: The XGRID Simulation Framework

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. While FPGAs do suffer increased power consumption as a result of large scale switching networks [26, 27], our proposed XGRID architecture has a low-cost static interconnect network, making it compile-time configurable with minimal power overhead.

3. 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 predictor 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, small (area-wise), and ultra 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 [6]. 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 interconnect is compile-time configurable.

An instance of the XGRID interconnection network with two rows and two columns is shown in Figure 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 Figure 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 as shown in Figure 2. 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 [7].

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.

4. THE XGRID SIMULATION FRAMEWORK

The XGRID simulation framework simulates the complete XGRID many-core system-on-chip (SoC) device, including processing cores, on-chip memories per core, interconnection network fabric, and serial high-speed I/O units. We implemented the simulation framework in C++, considering the hardware organization of the proposed XGRID architecture.

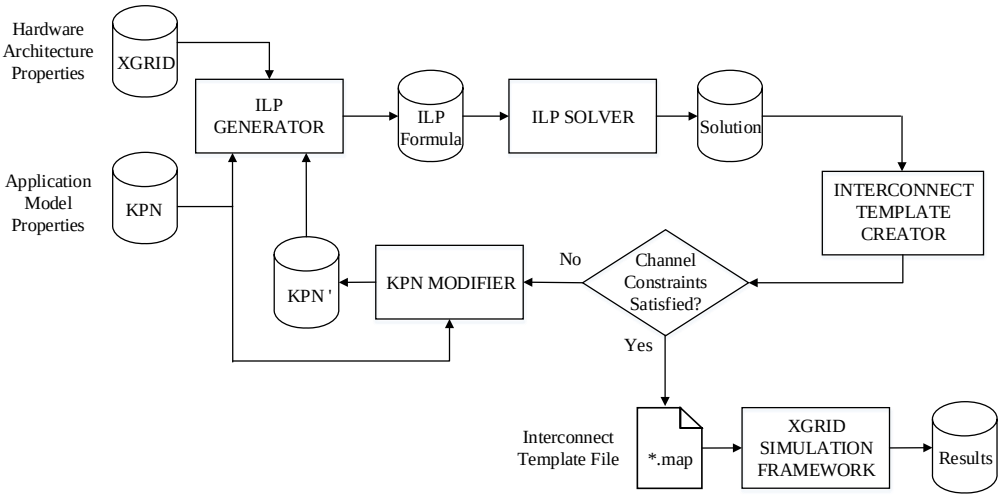


Figure 4: Holistic view of application mapping onto XGRID

As shown in Figure 3, the XGRID simulation framework consists of a complete software development tool-chain as well as a design verification tool-chain. The software flow is as follows. A sequential application is manually partitioned and revised for a multi-core target. The multi-core software application is fed through the XGRID compiler/assembler/linker tool-chain (i.e., tools based on the LCC [36]). The resulting object files are automatically mapped to cores on XGRID and the required communication channels are automatically established as needed. The mapping and routing of a parallel application to XGRID is automated using a heuristic algorithm (described later).

Once mapped to XGRID, the entire software/hardware system may be simulated to validate functional properties as well as obtain power/performance measurements. Specifically, the XGRID profiler provides per core statistics (e.g. the number of the instructions simulated, input/output rates, IO wait, compute time, computation and communication energy consumption) as well as chip-level dynamic attributes. We have developed a web-based user interface to drive the background design, simulation, and profiling tools in order to provide a fast and intuitive mechanism for students/developers to gain insight into the workings of parallel algorithms and many-core architectures.

5. APPLICATION MAPPING

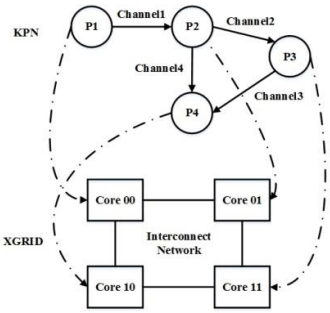
We follow a general scheme for our design flow of application mapping. This scheme includes application modeling and the actual mapping stages. Our holistic application mapping is shown in Figure 4. First of all, a sequential application is manually partitioned and represented as a Kahn Process Network (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 the interconnect of 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 generator follows the same flow by taking the modified KPN (KPN') as an input for the application model.

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. While in a pure KPN, processes are assumed to be deterministic (i.e., the same inputs always produce same outputs), we make no such assumptions, as processes running within our KPN models may maintain internal state. Each node of a KPN corresponds to one process (or task from the parallel algorithm). Communication 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. The unbounded nature of the KPN does not present a functional concern when mapped to bounded buffers of the XGRID. Instead, in XGRID the sending processes may be blocked, 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. The objective function, which we are aiming to minimize, is the overall communication cost of the system configuration. We used the CPLEX Optimization Studio 12.3 [22] for solving the set of integer linear equations.

The followings are the constraints and bounds, obtained from the KPN, to create an ILP for the ILP Solver:

Define → Formulate

Real Problem	Variables	Definitions	Constraints / Equations
	N	# of the rows in XGRID	$1 \leq N \leq P$
	M	# of the columns in XGRID	$1 \leq M \leq P$
	P	# of the processes in KPN	$P \leq N * M \leq 2 * P$
	N*M	# of the cores in XGRID	
	Process[*i*][0]	the row value of corresponding target core in XGRID for process *i* in KPN	$0 \leq Process[i][0] \leq (N-1)$
	Process[*i*][1]	the column value of corresponding target core in XGRID for process *i* in KPN	$0 \leq Process[i][1] \leq (M-1)$
	$Distance_{RD}[i]$	the distance between the target core for process *i* and an input unit.	$Distance_{RD}[i] = Process[i][1] + 1$
	$Cost_{RD}[i]$	the cost of reading data from an input unit for process *i*,	$Cost_{RD}[i] = Data_{IN}[i] * Distance_{RD}[i] * penalty\_off$
	penalty_off	a constant penalty for off-chip communication	
	$Data_{IN}[i]$	the number of read attempts from the input unit by process *i*,	
	$Distance_{WRT}[i]$	the distance between the target core for process *i* and an output unit.	$Distance_{WRT}[i] = N - Process[i][0]$
	$Cost_{WRT}[i]$	the cost of writing data to an output unit for process *i*	$Cost_{WRT}[i] = Data_{OUT}[i] * Distance_{WRT}[i] * penalty\_off$
	$Data_{OUT}[i]$	the number of write attempts to the serial output unit by process *i*	
	$Distance_{COM}[i][j]$	the distance between the core that process *i* is mapped to and the core that process *j* is mapped to	$Distance_{COM}[i][j] = abs(Process[i][0] - Process[j][0]) + abs(Process[i][1] - Process[j][1])$
	$Cost_{COM}[i][j]$	the cost of on-chip communication between process *i* and process *j*	$Cost_{COM}[i][j] = Data_{CHIP}[i][j] * Distance_{COM}[i][j]$
	$Data_{CHIP}[i][j]$	the total number of data transfer attempts between process *i* and process *j*	
	$Cost_{TOTAL}$	the global communication cost of the system for a specific mapping solution	$Cost_{TOTAL} = \sum_{i=0}^{P-1} \left\{ Cost_{RD}[i] + Cost_{WRT}[i] + \sum_{j=0}^{P-1} Cost_{COM}[i][j] \right\}$

ILP Solver ←

Figure 5: Summary of ILP Formulation for Mapping Problem

$$1 \leq N \leq P$$

$$1 \leq M \leq P$$

where, N and M are the number of the rows and columns in XGRID, respectively and P is the number of the processes in the KPN.

$$P \leq N * M \leq 2 * P$$

The above gives bounds for the number of cores ($N*M$) in XGRID. For all of the followings, i and j are from 0 to P-1.

$$0 \leq Process[i][0] \leq (N-1)$$

$$0 \leq Process[i][1] \leq (M-1)$$

where, $Process[i][0]$ and $Process[i][1]$ holds row and column values of the target core for process i, respectively. We assume each KPN process is to be mapped onto a single core – the premise of XGRID is that cores are plentiful. The result of the following must be different for all processes.

$$Process[i][0] * M + Process[i][1]$$

We use the following equations to calculate a cost function representing communication overhead of the overall system for a mapping solution.

$$Distance_{RD}[i] = Process[i][1] + 1$$

where, $Distance_{RD}[i]$ gives the distance between the target core for process i and a serial input unit.

$$Cost_{RD}[i] = Data_{IN}[i] * Distance_{RD}[i] * penalty\_off$$

where, $Cost_{RD}[i]$ is the cost of reading data from a serial input unit for process i, $Data_{IN}[i]$ is the number of read attempts from the serial input unit by process i, *penalty_off* is a constant penalty for off-chip communication.

$$Distance_{WRT}[i] = N - Process[i][0]$$

where, $Distance_{WRT}[i]$ gives the distance between the target core for process i and a serial output unit.

$$Cost_{WRT}[i] = Data_{OUT}[i] * Distance_{WRT}[i] * penalty\_off$$

where, $Cost_{WRT}[i]$ is the cost of writing data to a serial output unit for process i, $Data_{OUT}[i]$ is the number of write attempts to the serial output unit by process i.

$$Distance_{COM}[i][j] = abs(Process[i][0] - Process[j][0]) + abs(Process[i][1] - Process[j][1])$$

where, $Distance_{COM}[i][j]$ gives the distance between the core that process i is mapped to and the core that process j is mapped to.

$$Cost_{COM}[i][j] = Data_{CHIP}[i][j] * Distance_{COM}[i][j]$$

where, $Cost_{COM}[i][j]$ is the cost of on-chip communication between process i and process j, and $Data_{CHIP}[i][j]$ is the total number of data transfer attempts between process i and process j.

$$Cost_{TOTAL} = \sum_{i=0}^{P-1} \left\{ Cost_{RD}[i] + Cost_{WRT}[i] + \sum_{j=0}^{P-1} Cost_{COM}[i][j] \right\}$$

where, $Cost_{TOTAL}$ is the global communication cost of the system for a specific mapping solution.

The ILP solver (CPLEX) minimizes this total cost function ($Cost_{TOTAL}$) based on the given constraints and equations. The ILP formulation is summarized in Figure 5.

6. SYSTEM ENERGY PROFILING

Energy consumption is one of the most important characteristics of an embedded system. To evaluate the total energy consumption of XGRID, we take into account both computation and communication energy consumption. The following subsections provide our system energy models.

6.1 Computation Energy

A measurement based instruction-level energy model is proposed in [23] by measuring instant current drawn by the target ARM embedded processor and integrating the data to get base energy consumptions. The authors validated their proposed energy model with the experiments, conforming up to 5% error [23].

To profile the energy consumption of XGRID, for each core, we follow an instruction-level energy estimation methodology. The instruction set of XGRID is separated into four categories, namely, arithmetic/logic, load/store, control, and floating point instructions. A constant is defined as a base energy consumption of each category. To calculate these constants, we took averages of base energy consumption of related instructions as described in [23]. Our computation energy model is shown below.

$$\mathcal{E}_{compute} = \sum_{i=0}^{N*M-1} \{N_{Arth}[i] * C_1 + N_{LdSt}[i] * C_2 + N_{Ctrl}[i] * C_3 + N_{Float}[i] * C_4\}$$

where, $\mathcal{E}_{compute}$ is the total energy consumption of a computation, $N*M$ is the number of cores in XGRID; $N_{Arth}[i]$ is the number of arithmetic/logic instructions for core i; $N_{LdSt}[i]$ is the number of load/store instructions for core i; $N_{Ctrl}[i]$ is the number of control instructions for core i; $N_{Float}[i]$ is the number of floating point instructions for core i; and C_1, C_2, C_3, and C_4 are pre-calculated constants, which can be obtained from gate-level measurements per [23] or instrumentation of a physical core. For our experimentations, and particular XGRID instance, we used 1.328 nJ, 2.368 nJ, 1.644 nJ, and 2.656 nJ for C_1, C_2, C_3, and C_4, respectively.

6.2 Communication Energy

In our XGRID framework, there are two types of communication, namely, communication among cores (on-chip communication), and communication between cores and serial high speed I/O units (off-chip communication). We model the total energy consumption of communication as the sum of the total energy consumption of on-chip communication and off-chip communication, as shown below.

$$\mathcal{E}_{commun} = \mathcal{E}_{on\_chip} + \mathcal{E}_{off\_chip}$$

where, $\mathcal{E}_{commun}$ is the total energy consumption of communication, $\mathcal{E}_{on\_chip}$ is the total energy consumption of on-chip communication, and $\mathcal{E}_{off\_chip}$ is the total energy consumption of off-chip communication.

$$\mathcal{E}_{on\_chip} = \sum_{i=0}^{N*M-1} \sum_{j=0}^{N*M-1} \{N_{on\_chip}[i][j] * N_{hop\_on}[i][j] * C_{on\_chip} * {V_{dd}}^2\}$$

where, $N_{on\_chip}[i][j]$ is the total number of on-chip data transfer attempts between core i and core j, $N_{hop\_on}[i][j]$ is the distance between two communicating cores (core i and core j), C_{on_chip} is the line capacitance of the channel for on-chip communication, and V_{dd} is the supply voltage. The distance between cores are calculated according to their row and column values (distance = Δrow + Δcolumn).

$$\mathcal{E}_{off\_chip} = \sum_{i=0}^{N*M-1} \{(N_{off\_chip\_in}[i] * N_{hop\_off\_in}[i] + N_{off\_chip\_out}[i] * N_{hop\_off\_out}[i]) * C_{off\_chip} * {V_{dd}}^2\}$$

where, $N_{off\_chip\_in}[i]$ is the number of read attempts from a serial input unit by core i, $N_{hop\_off\_in}[i]$ is the distance between core i and the serial input unit, $N_{off\_chip\_out}[i]$ is the number of write attempts to a serial output by core i, $N_{hop\_off\_out}[i]$ is the distance between core i and the serial output unit, C_{off_chip} is the line capacitance of the channel for off-chip communication, and V_{dd} is the supply voltage.

We assume that off-chip data transfers consume more energy than on-chip data transfers. Therefore, we use two distinct constants for the line capacitances. As described in [24], C_{on_chip} and C_{off_chip} are set to 1.1 pF and 10 pF, respectively and the supply voltage of XGRID is fixed at 2.7V. These numeric constants will depend on the particular target implementation of the cores and manufacturing technology.

7. 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 the 2D DCT (Discrete Cosine Transform) benchmark, MMUL (Matrix Multiplication) benchmark, 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 [25], HYPER-QSORT [25], and PSRS-QSORT [25].

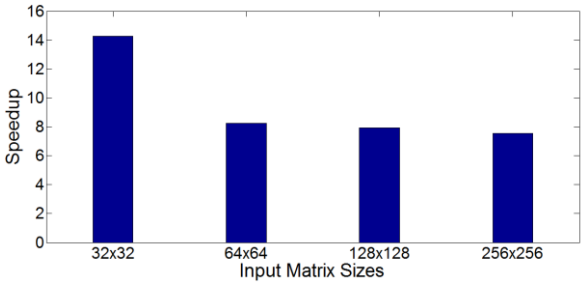


Figure 6: 2D DCT speedup (vs. single-core) on XGRID

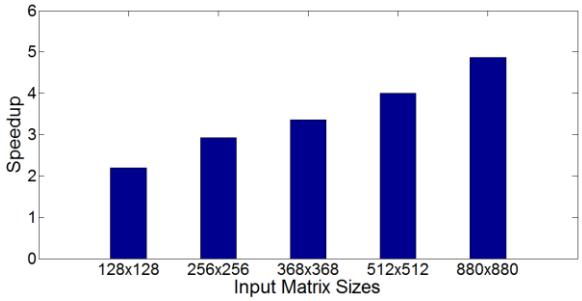


Figure 7: MMUL speedup (vs. single-core) on XGRID

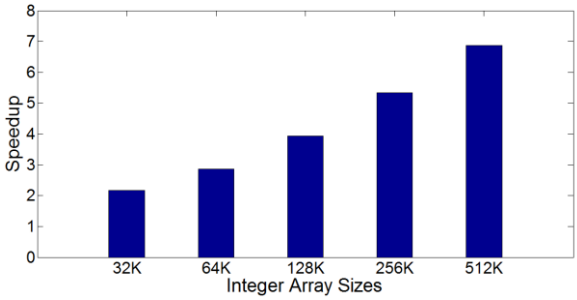


Figure 8: PSRS-QSORT speedup (vs. single-core) on XGRID

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 1MB data cache and a 64KB instruction cache.

Table 1 and 2 summarize our simulation results for the 2D DCT running on a single core and many-core, respectively. Table 3 and 4 summarize our simulation results for the MMUL running on a single core and many-core, respectively. Table 5 summarizes our simulation results for the QSORT benchmark running on a single core. Table 6, 7, and 8 summarize our simulation results for the three different parallel versions of QSORT, namely, PARALLEL-QSORT, HYPER-QSORT, and PSRS-QSORT running on many-cores.

Figure 6 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. Figure 7 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, as the initial cost of reading the matrices relative to the cost of multiplication diminished. Figure 8 shows the performance speedup of PSRS-QSORT relative to a single-core QSORT implementation. PSRS-QSORT offered the best performance improvement (average of 4X). As with MMUL, the performance improvement of PSRS-QSORT improved, as the size of the input array increased.

PARALLEL-QSORT benchmark shows poor performance, compared to HYPER-QSORT and PSRS-QSORT. This is not unexpected, as the latter two versions are optimized for many-core implementations. Execution time of PARALLEL-QSORT is bottlenecked by the last core finishing its execution; hence, the algorithm is likely to do a poor job balancing the number of elements sorted by each core [25]. HYPER-QSORT shows better performance than PARALLEL-QSORT, as expected. Here, the number of elements sorted by each core stays reasonably balanced. As the size of the array to be sorted increases, communication overhead limits scalability of the algorithm [25]. Communication time for HYPER-SORT is dominated by I/O wait. PSRS-QSORT is the best among our parallel algorithms based on QSORT. It does an excellent job balancing the number of elements sorted by each core [25] and I/O wait takes very little time. We included the results for PARALLEL-QSORT and HYPER-QSORT to 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. Figure 9 and Figure 10 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 matrix multiplication is a communication intense application therefore for increasing number of cores above a certain level, the communication time dominates the computation time and, as a result, causes a decrease in performance. Table 9 summarizes the results for XGRID varying in core count.

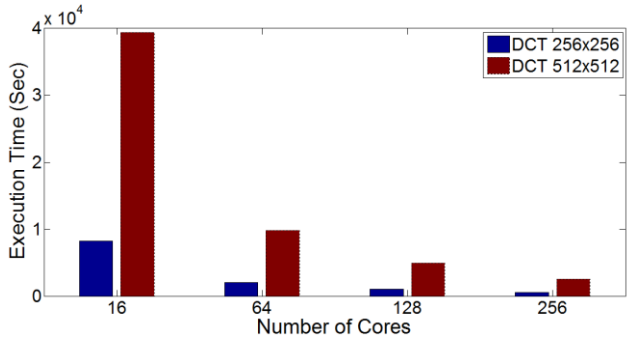


Figure 9: Execution time of 2D DCT benchmarks running on XGRID of various size (core count)

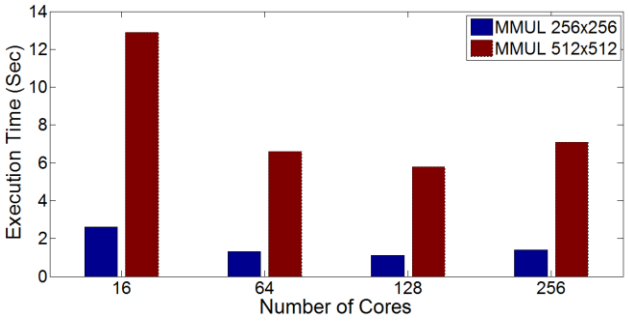


Figure 10: Execution time of MMUL benchmarks running on XGRID of various size (core count)

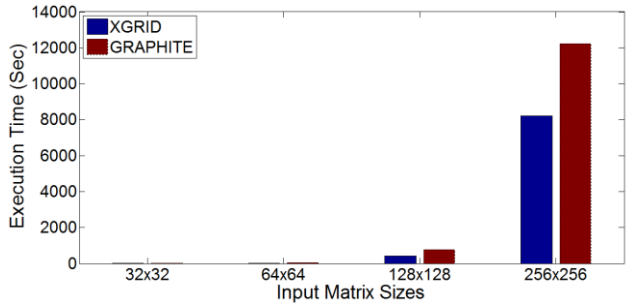


Figure 11: Execution time comparison of 2D DCT benchmarks (XGRID vs GRAPHITE using 16 cores)

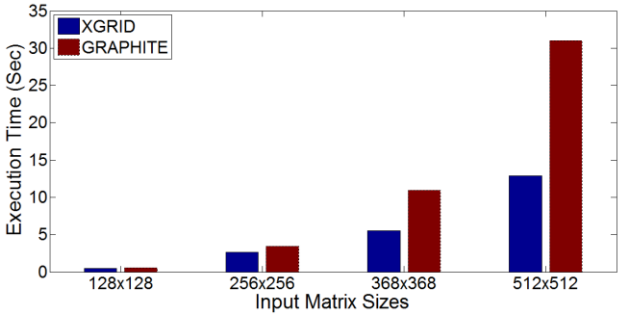


Figure 12: Execution time comparison of MMUL benchmarks (XGRID vs GRAPHITE using 16 cores)

We also validated our assertions on XGRID architecture by comparing XGRID against Graphite many-core architecture. Figure 11 and Figure 12 show the execution time comparison of 2D DCT and MMUL benchmarks, respectively. For the sake of fairness in comparison, we designated the features stated in Table 12 for both architectures. We ran some of our benchmarks on an open-source simulator for Graphite many-core architecture [35], 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 [32]. 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. Table 10 summarizes the results for the 2D DCT running on both 16-core XGRID and Graphite. Table 11 summarizes the results for the MMUL running on both 16-core XGRID and Graphite. The results show that XGRID outperformed Graphite in execution time in all the cases.

8. CONCLUSIONS

The transition to many-core architectures for embedded application is inevitable. 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.

Abbreviations: Exe. : Execution, Tot. : Total, Comm.: Communication, and Comp.: Computation

Table 1. Execution Time and Energy Consumption of 2D-DCT for Single Core

2D - DCT Matrix Sizes	Exe. Time (Sec)	Tot. Energy (J)	Tot. Energy of Comp. (J)	Tot. Energy of Comm. (µJ)
32 x 32	11.4	25.2	25.2	2.2
64 x 64	180	117	117	9
128 x 128	3337	250.2	250.2	36.4
256 x 256	61880	423	423	150

Table 2. Execution Time and Energy Consumption of 2D-DCT for 16 cores in XGRID

2D - DCT Matrix Sizes	Exe. Time (Sec)	Tot. Energy (J)	Tot. Energy of Comp. (J)	Tot. Energy of Comm. (µJ)
32 x 32	0.8	5.6	5.6	2.8
64 x 64	21.8	96.7	96.7	11.3
128 x 128	421	227.6	227.6	46
256 x 256	8210	256	256	188

Table 3. Execution Time and Energy Consumption of MMUL for Single Core

MMUL Matrix Sizes	Exe. Time (Sec)	Tot. Energy (J)	Tot. Energy of Comp (J)	Tot. Energy of Comm. (µJ)
128 x 128	1.1	1	1	35
256 x 256	7.3	7.5	7.5	144
368 x 368	18.5	20	20	287
512 x 512	51.3	53.5	53.5	584
880 x 880	245	62	62	1817

Table 4. Execution Time and Energy Consumption of MMUL for 16 cores in XGRID

MMUL Matrix Sizes	Exe. Time (Sec)	Tot. Energy (J)	Tot. Energy of Comp (J)	Tot. Energy of Comm. (µJ)
128 x 128	0.5	0.3	0.3	53
256 x 256	2.5	1.7	1.7	215.5
368 x 368	5.5	4.3	4.3	428.5
512 x 512	12.8	11.5	11.5	870
880 x 880	50.3	53.2	53.2	2674

Table 5. Execution Time and Energy Consumption of QSORT for Single Core

QSORT Integer Array Sizes	Exe. Time (Sec)	Tot. Energy (J)	Tot. Energy of Comp. (J)	Tot. Energy of Comm. (µJ)
32K	1.3	1.5	1.5	44.3
64K	4	4	4	89.5
128K	13	14	14	182
256K	46.4	49	49	372
512K	174	99.5	99.5	752

Table 6. Execution Time and Energy Consumption of PARALLEL-QSORT in case of using 16 cores in XGRID

PARALLEL-QSORT Integer Array Sizes	Exe. Time (Sec)	Tot. Energy (J)	Tot. Energy of Comp. (J)	Tot. Energy of Comm. (µJ)
32K	6.6	3.9	3.9	46.7
64K	33	18.5	18.5	94.4
128K	130	58.5	58.5	192
256K	512	113	113	391.5
512K	2041	190	190	791

Table 7. Execution Time and Energy Consumption of HYPER-QSORT in case of using 16 cores in XGRID

HYPER-QSORT Integer Array Sizes	Exe. Time (Sec)	Tot. Energy (J)	Tot. Energy of Comp. (J)	Tot. Energy of Comm. (µJ)
32K	1.8	2.7	2.7	46.7
64K	5.8	10	10	94.3
128K	20.6	39.6	39.6	191.6
256K	77.5	149.8	149.8	391
512K	299	220	220	790.5

Table 8. Execution Time and Energy Consumption of PSRS-QSORT in case of using 16 cores in XGRID

PSRS-QSORT Integer Array Sizes	Exe. Time (Sec)	Tot. Energy (J)	Tot. Energy of Comp. (J)	Tot. Energy of Comm. (µJ)
32K	0.6	0.3	0.3	47
64K	1.4	1	1	94.6
128K	3.3	3.1	3.1	192.3
256K	8.7	10.8	10.8	392.7
512K	25.3	40	40	793.4

Table 9. Execution Time of benchmarks running on XGRID of various sizes (core count).

Cores	256x256 2D DCT	512x512 2D DCT	256x256 MMUL	512x512 MMUL
	Exe. Time (Sec)	Exe. Time (Sec)	Exe. Time (Sec)	Exe. Time (Sec)
16	8210.0	39351.0	2.5	12.8
64	2059.0	9797.0	1.3	6.6
128	1033.0	4916.0	1.1	5.8
256	525.0	2497.0	1.4	7.1

Table 10. XGRID vs. GRAPHITE [32] using 2D DCT (on 16 cores)

Input Size	XGRID	GRAPHITE	Improvement
	Exe. Time (Sec)	Exe. Time (Sec)	
32x32	0.8	3.0	275%
64x64	21.8	47.6	118%
128x128	421.0	764.2	82%
256x256	8212.0	12236.0	49%

Table 11. XGRID vs. GRAPHITE [32] using MMUL (on 16 cores)

Input Size	XGRID	GRAPHITE	Improvement
	Exe. Time (Sec)	Exe. Time (Sec)	
128x128	0.5	0.54	8%
256x256	2.6	3.4	31%
368x368	5.5	10.9	98%
512x512	12.9	31.0	140%

Table 12. The designated values for some features of XGRID and GRAPHITE [32] architectures

Feature	XGRID	GRAPHITE
Number of Cores	16	16
Clock Frequency	130 Mhz	130 Mhz
Instruction Cache	Private, 64 KB (per core)	Private, 64 KB (per tile)
Data Cache	Private, 1 MB (per core)	Private, 1 MB (per tile)
Interconnect Topology	2D grid network	2D mesh network

9. 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] K. Skadron, et al. Temperature-Aware Microarchitecture. In *Proceedings of the 30th International Symposium on Computer Architecture*, June 2003.

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

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

[5] A. Baumann, et al. The Multikernel: A new OS architecture for scalable multicore systems. In *Proc. of ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 29–44, 2009.

[6] K.K.W Poon, S.J.E Wilton. A detailed power model for field programmable gate arrays. *ACM Trans. on Design Automation of Electronic Systems*, vol. 10, no. 2, pp. 279-302, April 2005.

[7] D. Walker. The design of a standard message passing interface for distributed memory concurrent computers. *Parallel Computing*, 20:657–673, 1994.

[8] S. Pasricha, N. Dutt. On-Chip Communication Architectures - System on Chip Interconnect © 2008 *Elsevier*

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] P. Guerrier and A. Grenier. A Generic Architecture for On-Chip Packet-Switched Interconnections. In *Proc. of Design, Automation, and Test in Europe (DATE) Conference*, pp. 250-256, 2000.

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

[15] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *IEEE Transactions on Computers*, vol. 54, no. 8, pp. 1025–1040, Aug. 2005.

[16] S. Patel, P. Parandkar, S. Katiyal and A. Agrawal. Exploring Alternative Topologies for Network-on-Chip Architectures**.** In *BVICAM's International Journal of Information Technology (BIJIT)*, vol. 3, issue. 2, August 2011.

[17] T. Simunic, S. P. Boyd, and P. Glynn. Managing Power Consumption in Networks on Chips. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 1, Jan. 2004.

[18] U. Y. Ogras, J. Hu, and R. Marculescu. Key research problems in NoC design: A holistic perspective. In *Proc. of International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 69–74, Sep. 2005.

[19] J. W. van den Brand, C. Ciordas, K. Goossens, and T. Basten. Congestion-Controlled Best-Effort Communication for Networks-on-Chip. In *Proc. of Design, Automation, and Test in Europe (DATE) Conference*, pp. 948–953, Apr. 2007.

[20] J. Cong, et al. MC-Sim: An Efficient Simulation Tool for MPSoC Designs. In *Proc. of IEEE/ACM Inter. Conference on Computer-Aided Design (ICCAD)*, pp. 364–371, 2008.

[21] G. Kahn. The Semantics of a Simple Language for Parallel Programming. In *Proc. of IFIP Congress*, Stockholm, 1974

[22] CPLEX Optimization Studio. [Online]. Available: http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/

[23] N. Kavvadias, et al. Measurements Analysis of the Software-Related Power Consumption in Microprocessors. In *Proc. of Instrumentation and Measurement Technology Conference (IMTC)*, 2003.

[24] A.B.A. Garcia, J. Gobert, T. Dombek, H. Mehrez and F. Petrot. Energy Estimation in High Level Cycle-Accurate Descriptions of Embedded Systems. In *Proc. of IEEE Inter. Conference on Electronics, Circuits and Systems*, 2002.

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

[26] E. Kusse, J.Rabaey. Low-Energy Embedded FPGA Structures. *International Symposium of Low Power Electronic Design (ISLPED)*, pp.155-160, 1998.

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

[28] N. L. Binkert, et al. The M5 Simulator: Modeling Networked Systems. *IEEE Micro*, vol.26, no.4, pp. 52-60, 2006.

[29] M.M.K. Martin, et al. Multifacet's General Execution-Driven Multiprocessor Simulator (GEMS) Toolset. *Sigarch Computer Architecture News*, vol. 33, no. 4, pp. 92-99, Sept. 2005.

[30] N. Binkert, et al. The gem5 Simulator. *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1-7, May 2011.

[31] P. Ren, et al. Hornet: A Cycle-Level Multicore Simulator. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 6, pp. 890-903, June 2012.

[32] 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.

[33] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, N. P. Jouppi. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. *42nd Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2009.

[34] 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.

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

[36] LCC, A Retargetable C Compiler. [Online]. Available: https://aur.archlinux.org/packages/lcc-compiler/