OpenSoC Fabric: On-Chip Network Generator

Using Chisel to Generate a Parameterizable On-Chip Interconnect Fabric

Farzad Fatollahi-Fard, David Donofrio, George Michelogiannakis, John Shalf
ffard@lbl.gov, ddonofrio@lbl.gov, mihelog@lbl.gov, jshalf@lbl.gov
Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, California 94720

ABSTRACT

Recent advancements in technology scaling have sparked a trend towards greater integration with large-scale chips containing thousands of processors connected to memories and other I/O devices using non-trivial network topologies. Software simulation suffers from long execution times or reduced accuracy in such complex systems, whereas hardware RTL development is too time-consuming. We present *OpenSoC Fabric*, a parameterizable and powerful on-chip network generator for evaluating future large-scape chip multiprocessors and SoCs. *OpenSoC Fabric* leverages a new hardware DSL, Chisel, which contains powerful abstractions provided by its base language, Scala, and generates both software (C++) and hardware (Verilog) models from a single code base. This is in contrast to other tools readily available which typically provide either software or hardware models, but not both. The *OpenSoC Fabric* infrastructure is modeled after existing state-of-the-art simulators, offers large and powerful collections of configuration options, is open-source, and uses object-oriented design and functional programming to make functionality extension as easy as possible.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Design

Keywords

Network-on-Chip, Emulation, Simulation, FPGA, Modeling, Chisel

NoCArC'14, December 13-14 2014, Cambridge, United Kingdom
ACM 978-1-4503-3064-0/14/12 ...$15.00
http://dx.doi.org/10.1145/2685342.2685351.

1. INTRODUCTION

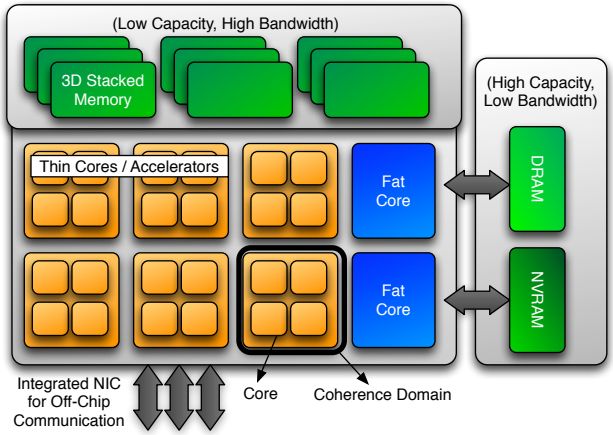

Figure 1: Abstract Machine Model of an exascale node architecture [4].

While technology scaling continues to enable greater chip densities, the leveling-off of clock frequencies has led to the creation of chips with higher core counts, making parallelism the key to greater performance in the future [10, 32]. For instance, the abstract machine model for potential exascale node architectures indicates that the number of cores per chip will be on the order of thousands [4]. In addition, Intel's straw-man exascale processor is expecting to have 2048 cores by 2018 in 7nm technology [9]. At the same time, the cost of data movement is quickly becoming the dominant power cost in chip designs. Current projections indicate that the energy cost of moving data will outweigh the cost of computing (FLOP) by 2018 even for modest on-chip distances [18, 32]. The impact of on-chip communication has been demonstrated even in older technologies, such as in the Intel Teraflop Chip which attributes 28% of its power to the on-chip network [19]. Also, the on-chip network can have a noticeable impact on application execution time with many applications and systems, which can even reach 60% of execution time in extreme cases [15, 31].

Power consumption will be an even greater concern in future technologies, owing to the end of Dennard scaling which limits scaling down voltage frequencies [17]. In addition, because conventional network topologies such as crossbars and buses do not scale at such high component count [14, 24], we need to focus on complex and novel on-chip network designs with particular emphasis on scalability. This creates a dire need to easily explore the design space and innovate

on future large-scale chips, which are infeasible today. This, however, requires capable infrastructure for simulation and modeling at such large scales.

There is a number of software simulators publicly available that are in wide use today [2, 3, 11, 21, 33]. However, while software models are faster to develop, they face scalability challenges. Even a 1000 MIPS (MegaInstructions Per Second) software simulator may have 1000× slowdown for a 1024 core chip with no precision sacrifices [13]. Past work has resorted to multithreaded execution [33] or approximation techniques such as slack simulation that relax timing dependencies within certain time windows [12]. While these techniques can speed up execution, they either force users to accept tradeoffs such as result accuracy, or they do not speed up execution sufficiently for rapid design space exploration.

Alternative approaches use hardware emulation in ASICs or FPGAs. This infrastructure can be provided either with open-source RTL [7], or by RTL generators [25, 28, 29]. Hardware emulation solves the speed concerns of software simulators. However, developing a strictly RTL-based implementation is much too labor intensive [23]. In addition, gaining visibility to internals of hardware modules for the purpose of collecting statistics can be tedious.

OpenSoC Fabric strikes a balance between these two approaches by leveraging a new hardware DSL: Chisel [6]. Chisel (Constructing Hardware In a Scala Embedded Language) provides software (C++) and hardware (Verilog) models from a single code base, as shown in Fig. 2. This is a unique advantage compared to other tools readily available that perform either software simulation or hardware emulation, but not both. Users wishing to evaluate a design in both software and hardware need to use and potentially modify multiple tools, and verify that the functionality between them is equivalent.

Chisel provides powerful abstractions which brings hardware design closer to functional programming, therefore significantly speeding up development. *OpenSoC Fabric* provides a powerful set of configuration options and implements both wormhole and virtual channel flow control using credits and pipelined routers. *OpenSoC Fabric* is also made available open source with a modular design and standardized interface between modules as well as at the endpoints, such that its functionality can be readily extended to fit research and industrial needs. In this paper, we provide an overview of the internal architecture, novelty, usage, and future work of *OpenSoC Fabric*. We also extend an invitation to the on-chip network community to use this tool as well as contribute to our development efforts. Latest updates, documentation, and support can be found at `http://www.opensocfabric.org`.

2. BACKGROUND

In this Section, we provide more details for Chisel and then discuss related work.

2.1 Chisel

Written in Scala, Chisel raises the level of hardware design abstraction by providing concepts including object orientation, inheritance, functional programming, parameterized types, structures of variables, and type inference [6]. Using these features, we are able to quickly design and develop different parameterized network-on-chip topologies and network modules for *OpenSoC Fabric*. In fact, development

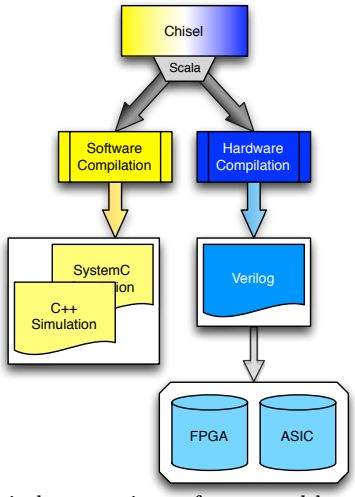

Figure 2: Chisel generating software and hardware from a single codebase.

time for added functionality is comparable to that of functional simulators (such as in C++), such that just a few hours to implement a flattened butterfly topology [22] based on a mesh topology code. Since it is based on Scala, Chisel can use the full power of Scala for non-synthesizable code, such as testbenches. Synthesizable code uses Chisel statements and modules, which includes a powerful library of often-used primitives, such as conversions between different encodings.

Chisel's software model is used for simple network design exploration with fast-functional models, while the hardware model can be used in emulation for cycle-accurate modeling. Using the synthesis path provided by Chisel, *OpenSoC Fabric* allows interesting and non-trivial network designs to be synthesized for a target ASIC process or FPGA flow, generating accurate power parameters and evaluating cycle time which software simulators are unable to do. These power measurements can then be back annotated to the software model to drastically improve the software model's power estimates. Alternatively, software models can collect their own statistics for the purposes of power estimation, and can even generate wave forms purely from functional simulation.

Chisel takes care of both the software and hardware models with one single codebase, as shown in Fig. 2. Given that developing Chisel code is much less tedious than RTL and comparable to functional programming, this means that the overall effort to construct both software and hardware models reduces dramatically. In addition, there is very little, if any, effort required to provably match the behavior of the software code to the RTL, which otherwise is a non-trivial and time-consuming process.

While the Chisel DSL is still in active development, there have been several examples of processors successfully manufactured that have been designed exclusively in Chisel [5, 6]. This illustrates the faithful representation of the Verilog that Chisel produces from the user's Scala description.

2.2 Related Work

A wide variety of software on-chip network simulators are publicly available, each with a different set of features and strengths [2, 3, 11, 21, 33]. In addition, many multiprocessor simulators include an on-chip network, or support integration with a dedicated on-chip network simulator [8, 20, 30].

Cycle-accurate simulators typically produce reasonably accurate results, even for detailed and time-sensitive interactions, and can be more readily verified against RTL [21]. However, cycle-accurate simulators get unacceptably slow for few hundred or thousands of network endpoints. In response, past work has applied multithreading [11, 33], event-driven execution [3], or approximation techniques such as slack execution [12]. While these techniques increase simulated cycles per second, they often tradeoff either accuracy or ease of development. Even so, many of these simulators struggle with designs at the scale of future large-scale chips. More importantly, these simulators provide purely software models. Therefore, one cannot use these, without a huge effort, to build an ASIC or emulate on an FPGA. This capability is important to more accurately evaluate power and area, as well as calculate the impact in cycle time of new mechanisms.

To produce hardware models, there are open-source implementations and RTL generators. Open-source implementations can be extended and reused, but do not provide software simulation models and can be challenging to extend to a powerful set of configuration options if based on Verilog or VHDL, because of the lack of mechanisms to easily configure a hierarchy of modules other than "ifdefs" [7]. In addition, most on-chip network generators are closed-source or application-specific. ARM's CoreLink Interconnect provides a network for ARM cores, but is specific to mobile applications and lacks any visibility into different network topologies [25]. Arteris FlexNoC is another commercial alternative but is closed-source [29], making collaborative, community-driven design space exploration difficult. Much like ARM CoreLink Interconnect, there's very limited visibility into the implementation details of the network and also target mobile applications [29]. Finally, CONNECT [28] is an academic RTL generator, but the generator code is also not provided open-source and therefore functionality cannot be readily extended.

Hardware emulation solves the speed concerns of software simulators. However, developing a comprehensive pure RTL implementation is much too labor intensive, especially for academic institutions [23]. In addition, gaining visibility to internals of hardware modules for the purpose of collecting statistics can be tedious.

The ability to use both software simulation and hardware emulation from a single code base that is comparable in development effort to C++ is what sets *OpenSoC Fabric* apart software simulators and hardware (RTL) generators. In addition, *OpensoC Fabric* is free to use and open-source, to ease modification and development by the broad community.

There has been past work on producing both software and hardware models from a single description to reducing duplicate efforts. C to Gates [16] and SystemC [27] are common ways the community goes around the pitfalls of traditional HDLs. However, these require high-level synthesis (HLS), which not only can be cost prohibitive, but also doesn't generate optimal RTL for ASICs. One could also only use the synthesizable subset of these languages, which ends up being no different from using traditional HDLs. Bluespec is also a common alternative used in the EDA community [26]. While Bluespec can provide great designer productivity when the task at hand matches the pattern encoded in the application programming model, they are a poor match for tasks outside their domain [6]. Another limiting factor is that Bluespec is not a free language like most other mainstream HDLs, which limits community collaboration, one of novel items about *OpenSoC Fabric*. Chisel addresses all of these issues and allows powerful abstractions and generators to be created. This underlying flexibility allows *OpenSoC Fabric* to accept a wide range of design parameters without losing the functionality provided by the object-oriented nature of Chisel.

3. OPENSOC FABRIC

As the name suggests, *OpenSoC Fabric* is the first step towards generating a configurable open-source system-on-chip (SoC). However, the on-chip networks generated by *OpenSoC Fabric* are not specific to any domain and can be used in a wide range of applications, from high performance computing to mobile applications. Also, given the open-source modular nature of *OpenSoC Fabric*, users can easily plug-and-play their own proprietary IP in place of our modules to better suit their specific application.

3.1 Endpoint Connectivity

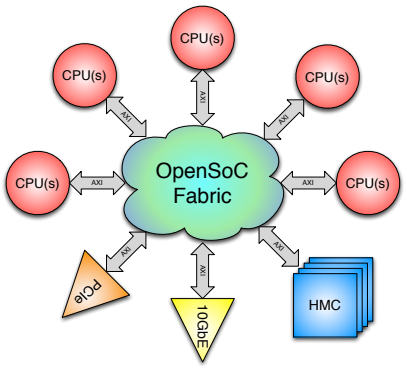

Figure 3: *OpenSoC Fabric* external connectivity with an example SoC configuration using AXI.

```
class MyOpenSoCFabric(parms: Parameters)
    extends OpenSoCFabric(parms) {
  val numPorts = parms.get[Int]("numPorts")
  val io = new Bundle { val Ports =
    Vec.fill(numPorts) (new AXI(parms)) }
  val myInterfaces = Vec.fill(numPorts)
    { Module (new NetworkInterface(parms)) }
  val myInjectionQs = Vec.fill(numPorts)
    { Module (new InjectionQ(parms)) }
  val myTopo = Module(new Topology(parms))
  val myEjectionQs = Vec.fill(numPorts)
    { Module (new EjectionQ(parms)) }
  for (i <- 0 until numPorts) {
    io.Ports(i) <> myInterfaces(i).io.AXIPort
    myInterfaces(i).io.OutChannels <>
      myInjectionQs(i).io.In
    myInjectionQs(i).io.Out <>
      myTopo.io.InChannels(i)
    myTopo.io.OutChannels(i) <>
      myEjectionQs(i).io.In
    myEjectionQs(i).io.Out <>
      myInterfaces(i).io.InChannels
  }
}
```

Listing 1: An example instantiation of *OpenSoC Fabric*. Detailed code and more examples can be found at `http://www.opensocfabric.org`.

One of the key qualities of integrating different IP blocks in SoCs or other large-scale chips is interoperability and abstraction. To this end, following object-oriented design, *OpenSoC Fabric* receives a configuration parameters object, but otherwise presents the same interface to external modules regardless of configuration. Currently, *OpenSoC Fabric* supports a simple ready/valid based interface that receives either packets or flits. To aid standardization but maintain simplicity, our future plans include developing a standardized AXI4-Lite interface [1], with possible future extensions to support additional features such as coherency support. This interface will accept entire messages. Thus, packetization will occur within the network boundaries before packets are injected into routers. Fig. 3 illustrates this concept for an example SoC configuration using AXI, and Lst. 1 shows the corresponding instantiation code in Chisel.

3.2 Internal Architecture

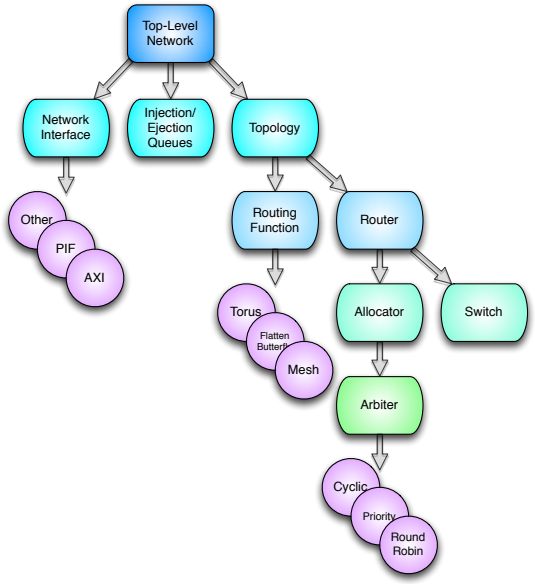

Figure 4: *OpenSoC Fabric* module hierarchy.

```
abstract class Allocator(parms: Parameters)
    extends Module(parms) {
  val numReqs = parms.get[Int]("numReqs")
  val numRes = parms.get[Int]("numRes")
  val arbCtor = parms.get[Parameters=>Arbiter]
    ("arbCtor")
  val io = new Bundle {
    val requests = Vec.fill(numRes)
      { Vec.fill(numReqs){ new RequestIO }.flip }
    val resources = Vec.fill(numRes)
      { new ResourceIO }
    val chosens = Vec.fill(numRes)
      { UInt(OUTPUT, Chisel.log2Up(numReqs)) }
  }
}
class SwitchAllocator(parms: Parameters)
    extends Allocator(parms) {
  // Implementation
}
```

Listing 2: Abstract modules define the common set of inputs and outputs for a standard interface.

We have designed *OpenSoC Fabric* to be hierarchical, modular, and highly parameterizable. As shown in Fig. 4, each *OpenSoC Fabric* module has a well-defined parent and set of children. For each module, there is an abstract implementation that includes the inputs, outputs, and functionality that is common to all child modules of that type. For example, the allocator abstract module defines the necessary input and output ports, as shown in Lst. 2. Specific allocator implementations, such as separable allocators, are defined as child modules that inherit those set of inputs and outputs, and implement the necessary functionality.

Every module inside *OpenSoC Fabric* is designed to have standard interface to other modules. This makes each module replaceable so users who want to do their own design space exploration and want to design their own modules are encouraged to do so. New modules can be individually developed and tested, and then included in the hierarchy. This is made even easier by using individual module testers already included in *OpenSoC Fabric*, like for allocators and routing functions. Finally, module configuration is performed by a configuration module instance passed down to child modules at instantiation time. When creating new modules, users need to add their configuration parameters to the configuration module, so that user options can be properly passed down the hierarchy.

While many of the above concepts that *OpenSoC Fabric*'s internal configuration stem from object-oriented design, applying these concepts to significantly ease development while producing hardware RTL in addition to software models is a new capability. *OpenSoC Fabric* makes it easy for users to configure the network as desired through the configuration class. *OpenSoC Fabric* also makes it easy for users who want to extend the network's functionality by editing code, through providing a hierarchical and modular design, as well as individual module testers.

4. CURRENT STATE AND FUTURE WORK

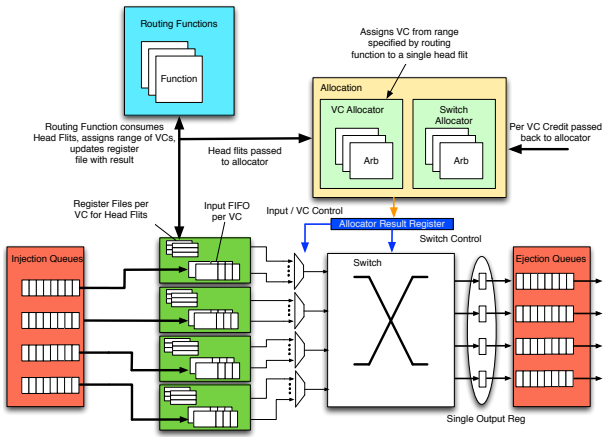

Figure 5: Block diagram of the implemented virtual channel router with injection and ejection queues at network boundaries.

OpenSoC Fabric currently supports both wormhole and virtual channel channel flow control with credits. A block diagram of the virtual channel router is shown in Fig. 5. Routers perform routing computation, switch traversal, and each allocation step in separate pipeline stages. Mesh and flattened butterfly [22] topologies are also supported, with

dimension-order routing and concentration. Also, we use separable input-first allocators and mux-based crossbars to implement the switch. *OpenSoC Fabric* includes a parameterizable test harness that can collect statistics and easily change the generated traffic pattern. Owing to its object-oriented design, extending the functionality of *OpenSoC Fabric* by—for example—implementing a new allocator only requires implementing a new child class with the new functionality. This is made easier by the included individual unit testers, which can be used to verify individual components in isolation from the network. Finally, *OpenSoC Fabric* includes some statistics collection capabilities, recording information such as router and channel utilizations as well as packet and flit latency and throughput statistics.

Our current plans for future work are as follows:

- *Validation:* We plan to validate *OpenSoC Fabric* against existing validated software models [21] or independently produced RTL [7]. We also plan to verify the functionality of the software models Chisel produces against the RTL. We have correlated our preliminary results with other tools as much as possible and have detected no discrepancies. However, we plan to perform a comprehensive validation against software and hardware models using synthetic and application traffic.
- *Different network technologies:* In it's current form, *OpenSoC Fabric* is focused on on-chip networks. However, we're already looking to move one level higher in the network hierarchy to apply *OpenSoC Fabric* to model internode communication as well as on-chip optical networks. This will help broaden the design-space exploration for future high performance computing.
- *Debugging and visualization:* We also would like to create a suite of tools to more easily debug designs, visualize the network, and produce power and area estimations in the software models.
- *Endpoint connectivity:* As previously stated, we are currently planning to implement the AXI4-Lite interface [1] at the network endpoints. As the list of features and use cases grows, we can extend the list of interfaces to include either custom or more powerful standardized interfaces.
- *More topologies and traffic patterns:* Finally, we plan to expand the collection of supported topologies, routing functions, and synthetic patterns.

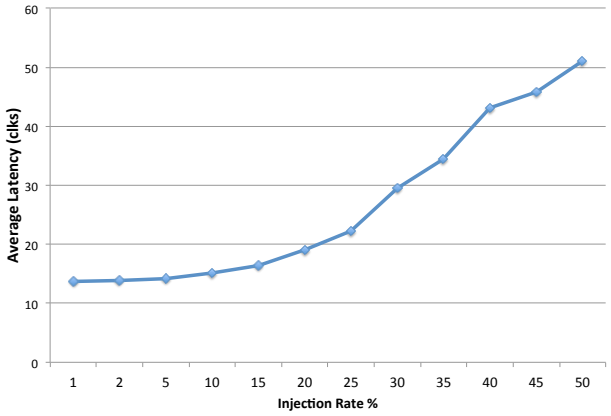

Figure 6: Injection rate and latency for a 4x4 mesh, concentration of one, and dimension order traffic.

4.1 Results

Using our current infrastructure, we have configured *OpenSoC Fabric* to generate an 4×4 mesh network with wormhole flow control and dimension order routing. Fig. 6 shows injection rate and latency for uniform random traffic and a concentration of one. Fig. 7 shows the latency of head-flits through the same network. Simulation speed for all our experiments is to the order of a few thousand network clock cycles per second on an AMD Opteron 8431 processor, comparable to other state of the art software simulators [2, 3, 11, 21, 33]. Through this process, we gained confidence in our ability to quickly generate both functional and RTL models and use them for software simulation and hardware emulation with little effort.

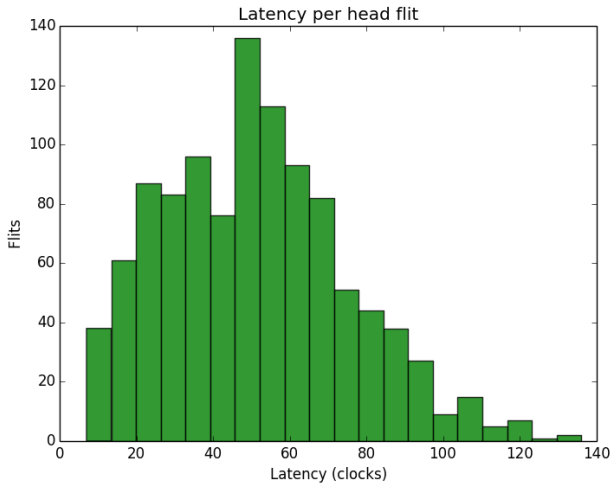

Figure 7: Latency of head flits through a 4x4 mesh concentration of one, and dimension order traffic.

5. CONCLUSIONS

In this paper, we presented *OpenSoC Fabic*. *OpenSoC Fabic* is a powerful on-chip network generator implemented in Chisel. This way, *OpenSoC Fabic* can generate both software (C++) and hardware (Verilog) models from a single code base. This is in contrast with other tools readily available which typically provide either software or hardware models, but not both. Chisel provides powerful abstractions which brings hardware design closer to functional programming, therefore significantly speeding up development. Therefore, significantly less effort is needed to create *OpenSoC Fabric* than building hardware and software models individually because these models are produced in parallel. *OpenSoC Fabric* provides a powerful set of configuration options with a modular design and standardized interfaces between modules as well as at the endpoints, such that its functionality can be readily extended to fit research and industrial needs. Being an open-source project, *OpenSoC Fabric* relies on collaborative effort. Therefore, we would like to extend an invitation to the community to use *OpenSoC Fabric* and then contribute back to aid tool growth.

References

[1] LogiCORE IP AXI4-Lite IPIF(v1.01.a). Technical report, Xilinx technical report, 2012. http://www.xilinx.com/support/documentation/ip_documentation/axi_lite_ipif_ds765.pdf.

[2] P. Abad, P. Prieto, L. Menezo, A. Colaso, V. Puente, and J.-A. Gregorio. Topaz: An open-source interconnection network simulator for chip multiprocessors and supercomputers. In *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, pages 99–106, May 2012.

[3] N. Agarwal, T. Krishna, L.-S. Peh, and N. Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 33–42, April 2009.

[4] J. Ang, R. Barrett, R. Benner, D. Burke, C. Chan, D. Donofrio, S. Hammond, K. Hemmert, S. Kelly, H. Le, V. Leung, D. Resnick, A. Rodrigues, J. Shalf, D. Stark, D. Unat, and N. Wright. Abstract machine models and proxy architectures for exascale computing. http://www.cal-design.org/publications/publications2, May 2014. Rev 1.1.

[5] K. Asanovic. Introduction and overview. Opening presentation at ASPIRE Retreat Summer 2014. "https://aspire.eecs.berkeley.edu/wiki/_media/retreats/2014summer/aspire-20140528-retreatintro.pptx", May 2014.

[6] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović. Chisel: Constructing hardware in a scala embedded language. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 1216–1225. ACM, 2012.

[7] D. U. Becker. *Efficient Microarchitecture for Network-on-Chip Routers*. PhD thesis, Stanford University, 2012. AAI0803043.

[8] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.

[9] S. Borkar. Exascale computing - a fact or a fiction? In *keynote presentation at IPDPS 2013*, May 2013.

[10] S. Borkar and A. A. Chien. The future of microprocessors. *Communications of the ACM*, 54(5):67–77, 2011.

[11] J. Chan, G. Hendry, A. Biberman, K. Bergman, and L. Carloni. Phoenixsim: A simulator for physical-layer analysis of chip-scale photonic interconnection networks. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 691–696, March 2010.

[12] J. Chen, M. Annavaram, and M. Dubois. Slacksim: A platform for parallel simulations of CMPs on CMPs. *SIGARCH Comput. Archit. News*, 37(2):20–29, July 2009.

[13] E. S. Chung, E. Nurvitadhi, J. C. Hoe, B. Falsafi, and K. Mai. A complexity-effective architecture for accelerating full-system multiprocessor simulations using fpgas. In *Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays*, FPGA '08, pages 77–86. ACM, 2008.

[14] W. J. Dally and B. Towles. Route packets, not wires: On-chip inteconnection networks. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, pages 684–689. ACM, 2001.

[15] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das. Application-aware prioritization mechanisms for on-chip networks. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 280–291. ACM, 2009.

[16] K. Denolf, S. Neuendorffer, and K. Vissers. Using c-to-gates to program streaming image processing kernels efficiently on fpgas. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 626–630, Aug 2009.

[17] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376. ACM, 2011.

[18] P. K. et al. Exascale computing study: Technology challenges in achieving exascale systems. http://users.ece.gatech.edu/~mrichard/ExascaleComputingStudyReports/exascale_final_report_100208.pdf, 2008.

[19] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. A 5-GHz mesh interconnect for a teraflops processor. *Micro, IEEE*, 27(5):51–61, 2007.

[20] C. L. Janssen, H. Adalsteinsson, S. Cranford, D. A. Evensky, J. P. Kenny, J. Mayo, and A. Pinar. A simulator for large-scale parallel computer architectures. *International Journal of Distributed Systems and Technologies*, 1(2):57–73, Apr. 2010.

[21] N. Jiang, D. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. Shaw, J. Kim, and W. Dally. A detailed and flexible cycle-accurate network-on-chip simulator. In *IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS, pages 86–96, 2013.

[22] J. Kim, W. J. Dally, and D. Abts. Flattened butterfly: a cost-efficient topology for high-radix networks. In *Proceedings of the 34th annual International Symposium on Computer Architecture*, ISCA '07, 2007.

[23] A. Krasnov, A. Schultz, J. Wawrzynek, G. Gibeling, and P.-Y. Droz. Ramp blue: A message-passing manycore system in fpgas. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 54–61, 2007.

[24] R. Kumar, V. Zyuban, and D. M. Tullsen. Interconnections in Multi-Core architectures: Understanding mechanisms, overheads and scaling. In *Proc. of the 32nd annual International Symposium on Computer Architecture*, pages 408–419, 2005.

[25] A. Ltd. Corelink interconnect - AMBA on-chip connectivity. http://www.arm.com/products/system-ip/interconnect/index.php, June 2014.

[26] R. S. Nikhil and Arvind. What is Bluespec? *SIGDA Newsl.*, 39(1):1–1, Jan. 2009.

[27] P. R. Panda. Systemc: A modeling platform supporting multiple design abstractions. In *Proceedings of the 14th International Symposium on Systems Synthesis*, ISSS '01, pages 75–80. ACM, 2001.

[28] M. K. Papamichael and J. C. Hoe. CONNECT: Re-examining conventional wisdom for designing nocs in the context of FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '12, pages 37–46. ACM, 2012.

[29] J. Probell. Playing well with others: How NoC compositions enable global team design. http://www.arteris.com/flexnoc, 2014.

[30] P. Ren, M. Lis, M. H. Cho, K. S. Shim, C. Fletcher, O. Khan, N. Zheng, and S. Devadas. Hornet: A cycle-level multicore simulator. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 31(6):890–903, June 2012.

[31] D. Sanchez, G. Michelogiannakis, and C. Kozyrakis. An analysis of interconnection networks for large scale chip-multiprocessors. *ACM Transactions on Architecture and Code Optimization*, 7(1):4:1–4:28, 2010.

[32] J. Shalf, S. S. Dosanjh, and J. Morrison. Exascale computing technology challenges. In *International Meeting on High Performance Computing for Computational Science*, volume 6449 of *VECPAR*, pages 1–25. Springer, 2010.

[33] A. T. Tran and B. Baas. NoCTweak: a highly parameterizable simulator for early exploration of performance and energy of networks on-chip. Technical Report ECE-VCL-2012-2, VLSI Computation Lab, ECE Department, University of California, Davis, 2012. http://www.ece.ucdavis.edu/vcl/pubs/2012.07.techreport.noctweak/.