A FPGA-Pipelined Approach for Accelerated Discrete-Event Simulation of HPC Systems

Carlo Pascoe, Sai P. Chenna, Greg Stitt, Herman Lam
PSAAPII Center for Compressible Multiphase Turbulence (CCMT)
NSF Center for High-Performance Reconfigurable Computing (CHREC)
Dept. of ECE, University of Florida, Gainesville FL 32608, USA
{pascoe, chenna, gstitt, hlam}@hcs.ufl.edu

EXTENDED ABSTRACT

In recent decades, HPC advancement has relied upon incremental improvements in commodity off the shelf components combined with large-scale integration. Looking forward, many practical challenges (e.g., power and memory cost, reliability) have necessitated a fundamentally different approach to HPC architectures and application development. Achieving exascale will require co-design, where application developers work closely with domain scientists and system architects to iteratively perform design-space exploration (DSE) to optimize algorithms on proposed systems [1]. Modeling and simulation of application behavior on target architectures is a key tool used in the co-design process. Recent trends in HPC system modeling and simulation suggest discrete-event simulation (DES) coupled with a multiscale approach to the modeling of the system and application behaviors may provide a good balance between model speed and accuracy [2, 4].

Although a multiscale approach enables faster simulation than traditional cycle-accurate approaches, exascale simulation with existing tools could take minutes, hours, or even days to complete a single simulation. These lengthy simulations place very practical limits on DSE and Uncertainty Quantification (UQ) efforts that often require thousands, or even millions of independent simulations. To address this issue, we propose FPGA-pipelined DES, which focuses not necessarily on improved performance for a single simulation, but instead on increased simulation throughput. By focusing on throughput, we unlock the potential for huge performance gains when the problem under study calls for numerous independent simulations (e.g., DSE, Monte Carlo simulation).

In our approach, we designed a custom compiler to convert the MPI parallel application and architecture specification used as input to existing simulation tools (Figure 1(a)) to a data-flow graph (DFG) representation (Figure 1(b)). Graph vertices represent each unique discrete event (e.g., matrix multiply, MPI send, barrier) and edges their input/output dependencies. The compiler utilizes several graph optimization techniques to manipulate the DFG before ultimately mapping it to an FPGA pipeline (Figure 1(c)). Assuming sufficient resources, the compiler simply maps each vertex operation of the DFG to independent FPGA resources. By adding pipeline registers between events with dependencies in the DFG, a single simulation has a latency equivalent to the DFG's critical path. More importantly, successive simulations start/complete once every cycle. However, if required resources exceed a single FPGA (a near certainty for exascale simulation) some degree of resource sharing

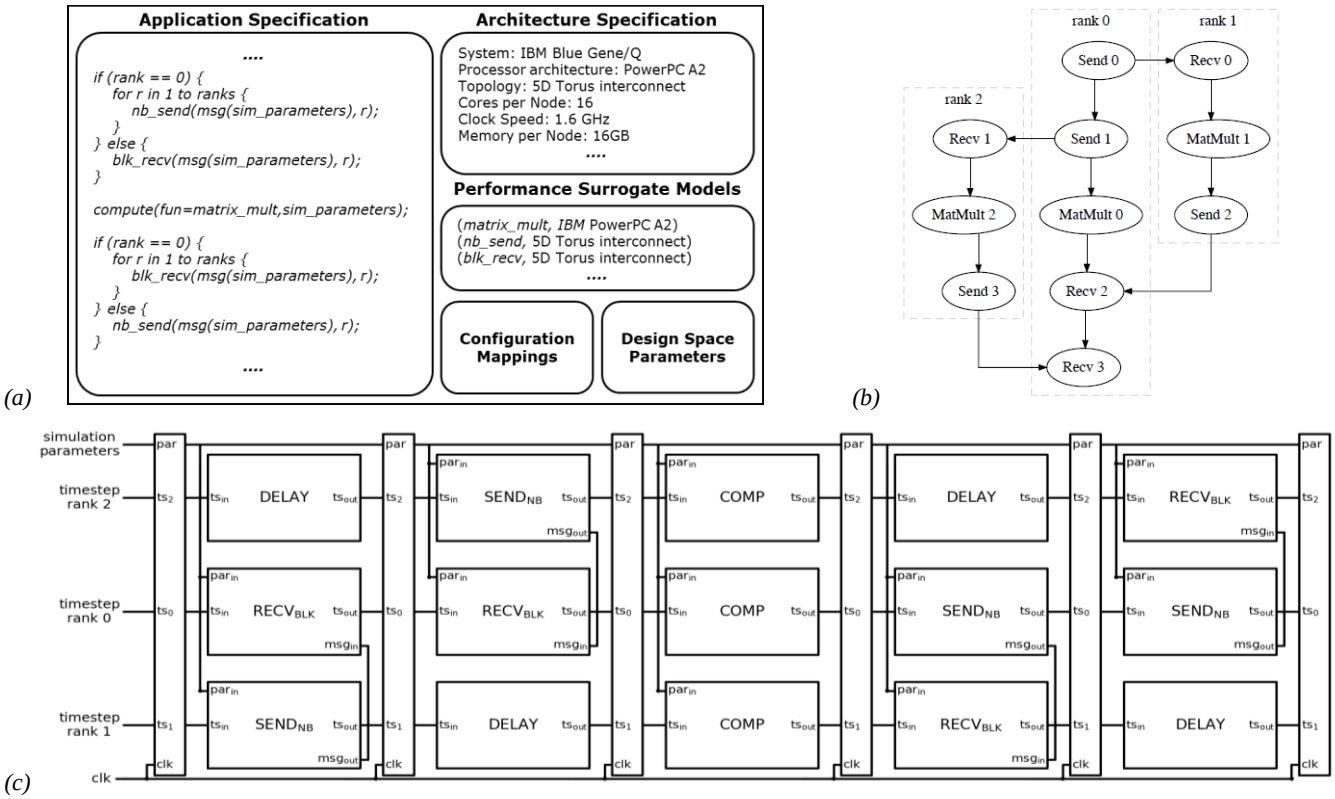

Figure 1. Example translation from (a) MPI-like simulation specification to (b) example data-flow graph and resulting (c) FPGA pipeline

Table 1. Performance of sharing (right) & no-sharing (left) pipelines for CMT-Bone-BE* with varied MPI ranks and simulation timesteps (TS) on a single Stratix V S5GSMD8K1F40C2. Performance based on average circuit speed of 335 MHz for sharing & 300 MHz for no-sharing pipelines. BE-SST [3] run on single Intel i7 core at 2.6 GHz.

	Ranks	TS	Num. of Events	% LU	Latency (cycles)	Hardware MSPS†	Hardware GEPS‡	BE-SST KEPS‡	Hardware Speedup
1.	32	1	1,344	15 / 2	64 / 278	300 / 10.5	403 / 14.1	4.4	$92x10^6$ / $3x10^6$
2.	32	2	2,688	31 / 3	118 / 512	300 / 10.5	806 / 28.2	7.8	$103x10^6$ / $4x10^6$
3.	32	3	4,032	46 / 4	172 / 746	300 / 10.5	1,210 / 42.3	10.6	$114x10^6$ / $4x10^6$
4.	32	4	5,376	61 / 6	226 / 980	300 / 10.5	1,613 / 56.4	12.9	$125x10^6$ / $4x10^6$
5.	32	5	6,720	76 / 7	280 / 1,214	300 / 10.5	2,016 / 70.6	14.8	$136x10^6$ / $5x10^6$
6.	32	6	8,064	92 / 9	334 / 1,488	300 / 10.5	2,419 / 84.7	16.5	$147x10^6$ / $5x10^6$
7.	32	8	10,752	– / 12	– / 1,916	– / 10.5	– / 113	19.1	– / $6x10^6$
8.	32	16	21,504	– / 24	– / 3,788	– / 10.5	– / 226	24.2	– / $9x10^6$
9.	32	32	43,008	– / 44	– / 7,532	– / 10.5	– / 452	28.9	– / $16x10^6$
10.	64	1	2,880	32 / 2	65 / 394	300 / 5.23	864 / 15.1	7.7	$112x10^6$ / $2x10^6$
11.	64	2	5,760	65 / 4	119 / 712	300 / 5.23	1,728 / 30.1	12.6	$137x10^6$ / $2x10^6$
12.	64	3	8,640	99 / 5	173 / 1,030	300 / 5.23	2,592 / 45.2	16.2	$160x10^6$ / $3x10^6$
13.	64	4	11,520	– / 7	– / 1,348	– / 5.23	– / 60.2	17.9	– / $3x10^6$
14.	64	8	23,040	– / 14	– / 2,620	– / 5.23	– / 121	23.2	– / $5x10^6$
15.	64	16	46,080	– / 29	– / 5,164	– / 5.23	– / 241	26.9	– / $9x10^6$
16.	64	32	92,160	– / 46	– / 10,252	– / 5.23	– / 482	29.1	– / $17x10^6$
17.	128	1	5,952	66 / 2	66 / 458	300 / 2.62	1,786 / 15.6	10.8	$165x10^6$ / $1x10^6$
18.	128	2	11,904	– / 4	– / 776	– / 2.62	– / 31.2	14.9	– / $2x10^6$
19.	128	3	17,856	– / 5	– / 1,094	– / 2.62	– / 46.8	17.1	– / $3x10^6$
20.	128	4	23,808	– / 7	– / 1,412	– / 2.62	– / 62.4	18.4	– / $3x10^6$
21.	128	8	47,616	– / 15	– / 2,684	– / 2.62	– / 125	20.8	– / $6x10^6$
22.	128	16	95,232	– / 30	– / 5,228	– / 2.62	– / 250	22.3	– / $11x10^6$
23.	128	32	190,464	– / 47	– / 10,316	– / 2.62	– / 499	22.8	– / $22x10^6$

***proxy app for CMT-Nek code under development at Florida's PSAAPII CCMT, †Mega-Simulations-Per-Second, ‡Giga/Kila-Events-Per-Second, '–' indicates configuration unable to fit on a single FPGA**

and/or circuit partitioning across multiple FPGAs is required. There exist many approaches to this problem (the focus of current and future research) and the challenge becomes how to apply them to automatically generate Pareto-optimal circuits for any simulation. We have developed a resource-sharing strategy that attempts to "collapse" the essentially 2D DFG (threads by events per thread) into a 1D pipe with two major scaling advantages: (1) resources scale sublinearly with the number of threads (with a factor of "threads" cost in simulation throughput), and (2) pipelines scale as a single, unidirectional pipe that can be partitioned predictably across any number of connected FPGAs with only minimal overhead.

Table 1 presents performance data for the two approaches while highlighting the advantages and disadvantages of each. The no-sharing approach clearly has superior performance in terms of simulation throughput and latency, but consumes far more resources and is ultimately far less scalable. Comparing lines 1-9 with lines 10-16 & 17-23, resources scale linearly with timesteps (TS) for both approaches, but the sharing approach allows for many more TS due to its much lower base utilization. Additionally, the sharing approach's resource utilization scales sublinearly with the number of threads (1, 10, & 17) allowing for simulations far larger than previously possible with the no-sharing approach on a single FPGA (e.g., simulated configurations up to 2,147,483,648 ranks). Although the sharing approach appears to suffer a significant drop in performance when considering simulation throughput (inverse proportionality to ranks), if we instead consider event throughput we see direct proportionality to logic utilization (LU%) almost independent of the number of ranks; simulation throughput decreases as the amount of work per simulation increases (increased events per simulation as number of ranks increase), but the amount of work completed each clock cycle remains constant and depends upon how much event hardware is instantiated. This ultimately means that although simulation throughput will continue to decrease with increased ranks, event throughput will remain relatively constant dependent on LU.

Overall, the proposed approach provides simulation/event throughput that is many orders-of-magnitude faster than the BE-SST software simulator [3] (speedup column shows at least 6 orders), however, this gain in throughput comes at a cost. One notable limitation of the FPGA-pipelined approach is a sacrifice in analysis capabilities largely due to limited I/O bandwidth (e.g., BE-SST can log all intermediate event data for postmortem analysis while the pipelined approach is limited to a handful of "monitored" events that can be logged without causing pipeline stalls). We also note that there are several potential application behaviors that the FPGA may not be able to efficiently handle at the same level of granularity as software (e.g., dynamically modifying control flow based on event timing). One possible solution is to model the application at a higher level of abstraction such that the behaviors are no longer present. For our envisioned use case, these limitations are not prohibitive because a designer can use our approach to rapidly prune a huge design space into a small set of promising candidates that can then be explored in more depth using existing techniques.

REFERENCES

[1] R.F. Barrett *et al.*, "On the role of co-design in high performance computing." Transition of HPC Towards Exascale Computing, Nov. 2013, pp 141–155.

[2] N. Kumar et al., "Behavioral Emulation for Scalable Design-Space Exploration of Algorithms and Architectures." E-MuCoCoS16.

[3] A. Ramaswamy et al., "Scalable Behavioral Emulation of Extreme-Scale Systems Using Structural Simulation Toolkit." (in preparation).

[4] A.F. Rodrigues *et al.*, "The structural simulation toolkit." ACM SIGMETRICS Performance Evaluation Review 38(4) 2011, pp 37–42.