Demonstrating Scalability of the Checkerboard GPC with SystemC TLM-2.0

Yutong Wang, Arya Daroui, and Rainer Dömer

CECS, University of California Irvine

Abstract. With the growing complexity of embedded applications, system architects integrate more processors into System-on-Chip (SoC) designs. Since scalability of such systems is a key criterion for their efficiency, regular array-type architectures are preferred that can easily grow in size. In this work, we model in SystemC TLM-2.0 a Grid of Processing Cells (GPC) with a Checkerboard arrangement of processors and memories. To demonstrate its scalability, we evaluate the performance of a highly parallel Mandelbrot renderer on growing Checkerboard platforms. Our results confirm that the performance scales well with the number of processors.

Keywords: SystemC · Scalability · System-on-Chip Design · TLM-2.0 · Grid of Processing Cells (GPC).

1 Introduction

The Grid of Processing Cells (GPC) has been proposed as a regular system architecture of many cores with local memories that are arranged in a scalable 2-dimensional array with only local interconnect [6]. The "Checkerboard" variant places memories in between processing cores in alternating fashion, allowing processors to access only neighboring memories. While the proposed GPC platform intuitively appears scalable, its scalability has not actually been shown. In this work [19], we design a detailed Checkerboard GPC model and describe it in SystemC TLM-2.0 [2]. Our model is fully functional, scalable in width and height, and can accurately simulate timing and thus measure performance. For our performance and scalability analysis, we choose the visualization of the Mandelbrot set [14] as a suitable application, because it is a perfectly (embarrassingly) parallel program, and map it onto the processing cells of the Checkerboard GPC. In addition to detailed timing measurements with growing Checkerboard sizes, we also compare the performance to a theoretical model with perfect linear scalability to show that the Checkerboard model also scales well.

1.1 Background and Related Work

Many computer architectures have been proposed and used throughout the years. The classic von Neumann computer architecture [11] has one memory bus between the memory and the central processing unit (CPU). The original Harvard

architecture [9] and its modern implementation, the modified Harvard architecture, all use a single shared memory bus. While modern computers are typically organized as symmetric multiprocessors (SMPs) [12], there is only a single shared memory connected via a bus interface. Having a single memory with a shared bus for CPU(s) severely limits the scalability of these architectures.

Knowing that the architectures with a single memory bus have a memory bottleneck, other architectures with better scalability have been proposed. The Raw architecture [15] is a 4x4 tiled architecture designed with multiple buses and multiple memories. It allows application-specific resource allocation and data flow within the chip. The tiled architecture of Raw also allows it to scale with increasing silicon density [15]. Another scalable architecture is the Tile Processor [20] [4]. The TILE64 and the TILEPro64 processor are both manufactured by Tilera. Both processors show the scalability of the tiled architecture. With each tile containing a general-purpose processor, a cache, and a router, TILE64 and TILEPro64 are able to communicate with each other and other I/O devices on a large 8x8 scale [16] [17] [1]. Intel's Teraflops Research Chip, codenamed Polaris, is another scalable many-core design with a network-on-chip architecture [18]. Polaris consists of a 10x8 2D mesh network (80 cores) with a sustained performance of 1.28 teraFLOPS, demonstrating very good scalability [13]. Intel's Single-Chip Cloud Computer (SCC) is another tiled platform that communicates through an architecture similar to a cloud computer in a data center. The chip contains tiles in a 4x6 2D-mesh with 2 P54C Pentium cores and a router in each tile. Intel hopes to make SCC scale to 100+ cores by having each chip communicate with another chip [8] [10]. KiloCore processor array, which contains 1000 independent processors and 12 memory modules on a single chip, is another scalable tiled-like architecture with multiple memory buses [3]. Their data indicates that under most conditions, the processor array has a near-optimal proportional scaling of power dissipation.

While the aforementioned related tiled architectures offer a large degree of scalability, the Checkerboard GPC studied in this work [19] promises to scale better in the sense that each processing core has only access to local/neighboring memories (which restricts application size and poses a burden on programmability). Thus, the memory access speed is not influenced by the size of the grid. Regardless of where a core is located in the grid, it has constant access to its neighboring memories, making Checkerboard GPC truly scalable.

1.2 Problem Definition

The Checkerboard Grid of Processing Cells has been proposed in [6] but not implemented. There is a need for a simulatable model to demonstrate the viability of the proposed architecture. In this work [19], we design a SystemC TLM-2.0 model and map a perfectly parallel application to it. We compare our experimental results to a theoretical model to show that the Checkerboard GPC truly scales well.

2 The "Checkerboard" Grid of Processing Cells

In this section, we review the Checkerboard GPC [6] and present its implementation in SystemC TLM-2.0.

2.1 Overview of Checkerboard SystemC Model

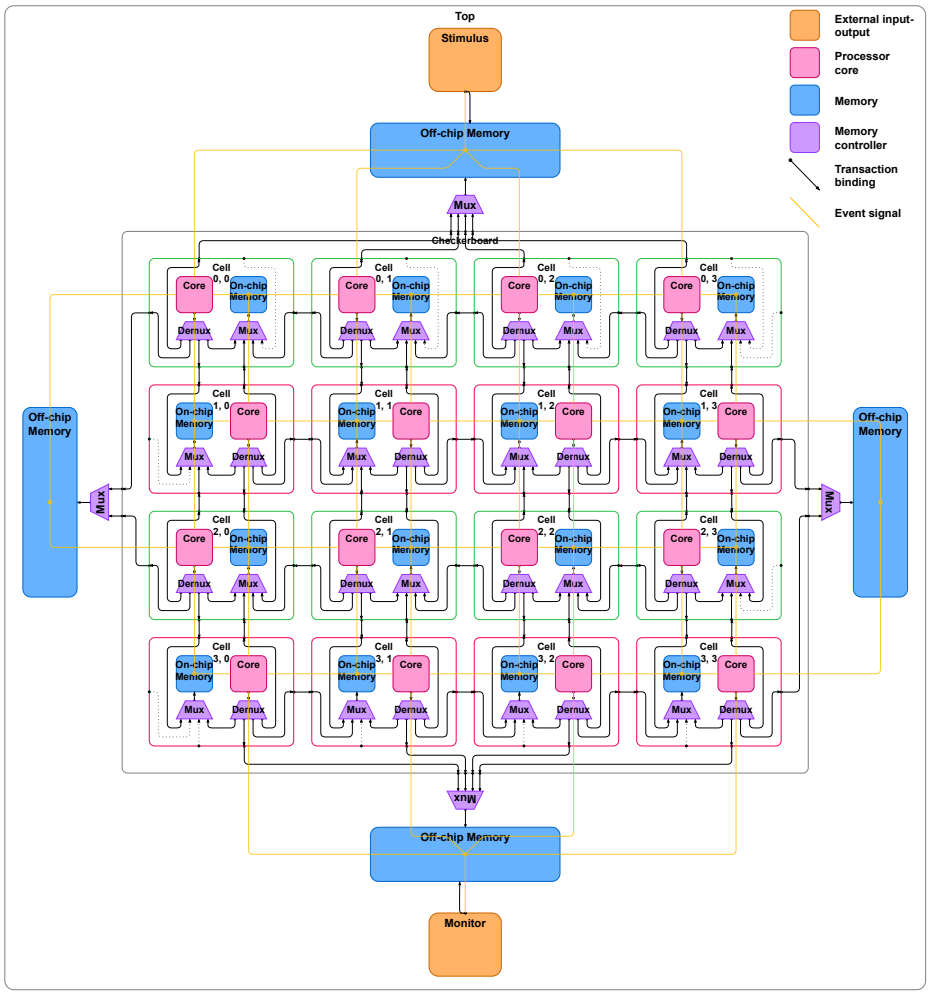

Fig. 1. Schematic of Checkerboard 4x4 GPC Model

Checkerboard Model Components An example Checkerboard 4x4 model schematic is shown in Figure 1. Our Checkerboard SystemC model contains many TLM-2.0 modules. The highest level module is named *Top*, and contains modules *Stimulus, Monitor, Off-Chip Memories, Multiplexers*, and *Checkerboard*. The *Checkerboard* module contains width by height *Cell* modules. Each *Cell* contains four modules, *Core, On-Chip Memory, Demultiplexer* and *Multiplexer*.

Checkerboard Model Configuration In the *Top* module, modules *Stimulus, Monitor, Multiplexers*, and *Off-Chip Memories* are configurable. In *Checkerboard* module, the number of *Cells* can be configured based on need with two parameters, Grid Width and Grid Height. We built a Python-based code generator that takes Grid Width and Grid Height and generates the corresponding SystemC code [19]. The user can also limit the size of both *On-Chip* and *Off-Chip Memories*. In module *Cell*, each *Core* module contains a SystemC thread, where the user provides the code for each *Core*. Timing (delay) for *On-Chip Memories*, *Off-Chip Memories*, and *Multiplexers* is also configurable.

Functionalities of Checkerboard Modules Module *Top* is a container for other modules. Inside *Top*, if the user does not provide their customized *Off-Chip Memory*, a default one-port memory is provided with basic read and write functions. The same rule applies to *Monitor* and *Stimulus*. All *Off-Chip Memories* in *Top* use TLM-2.0 standard sockets, shown as blue rectangles on the outside of *Checkerboard* module in Figure 1.

The *Multiplexers* inside *Top* contain sockets for connecting *Checkerboard*'s border cells to *Off-Chip Memories*. The purpose of these *Multiplexers* is to route incoming read and write requests from the *Cells* to the connected *Off-Chip Memory*. The *Off-Chip Multiplexers* are automatically configured.

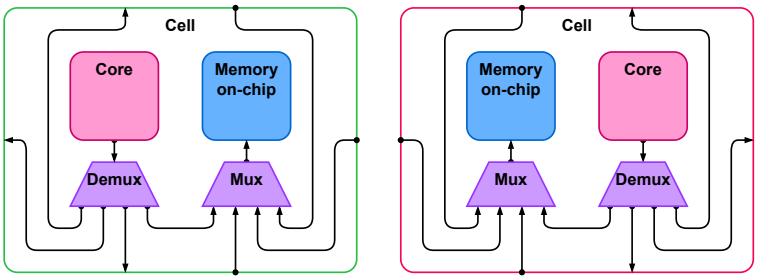

Fig. 2. Schematic of Cells Inside Checkerboard

The *Checkerboard* module contains a user-defined number of *Cells*. Each *Cell*, marked as green or red boxes in Figure 1, encloses 1 *Core*, 1 *Demultiplexer* (*Demux)*, 1 *On-Chip Memory*, and 1 *Multiplexer* (*Mux*), as shown in Figure 2. The *Checkerboard* contains 2 types of *Cells*: one with the *Core* module and *Core Demux* on the left side and *Memory* and *Mem Mux* on the right side; the other type is the opposite. Having 2 types of cell layouts allows each *Core* to have access to 4 adjacent memories without crossing wires. A *Core* has only 1 socket, and it is connected to a *Demux*. The *Core's Demux* takes the request from that *Core* and forwards it to the 4 connected memories based on the address of the *Core* and the address of the payload. Each *Core* has its own SystemC thread. The user is in charge of providing the functionalities of each thread.

Core-Memory-Core Communication The yellow lines that go across *Cells* in Figure 1 are SystemC event signals used to notify nearby *Cores* about memory

accesses. There is one event associated with each *On-Chip* and *Off-Chip memory*. This memory-event setup allows each *Core* to notify the events of neighboring memories and wake up nearby *Cores*, thus allowing safely synchronized *Core* to *Core* message passing.

Checkerboard Address Space As described in Section 2.1, when a *Core* sends a read or write request to a specific memory address, the payload goes through the *Core*'s *Demux*. The *Demux* forwards the payload to the correct memory based on the requested address and the sender's address.

Bit	31	Address bits	Address bits	31 - 2xAddress bits
On-chip memory	0	lg(height)	lg(width)	rest of address
Off-chip memory	1	pos	rest of address	

Fig. 3. Memory Address for Checkerboard Model

As shown in Figure 3, the most significant address bit is used to differentiate the *On-Chip* and *Off-Chip Memories*. For *Off-Chip Memories*, two more bits are needed to identify the four *Off-Chip Memories*. Address bits for *On-Chip Memories* depend on the number of bits needed to represent Grid Height and Grid Width of the Checkerboard. Each *Core* only uses its local address space, so the Checkerboard can truly scale to any size.

Model Limitations The current version of Checkerboard uses a 32-bit address space. As Checkerboard grid grows in size, each *On-Chip Memory* has a smaller maximum size. This also limits the current Checkerboard to a maximum size of 16x16 *Cells*. In the future, we design a 64-bit address space or higher to allow even larger Checkerboards. The mapping of any application onto the Checkerboard is currently done manually, which is prone to human error and also time-consuming. We plan to automate this process.

3 Mandelbrot Set Visualization on Checkerboard

We now describe the parallel application chosen to demonstrate scalability. For a detailed definition of the Mandelbrot Set, please refer to the original documentation [14].

3.1 Theoretical model of Mandelbrot Set Visualization

As a perfectly scalable reference we also build a Theoretical Mandelbrot Set Visualization model, theoretical model for short. The theoretical model is implemented with SystemC TLM-2.0 with accurate memory behavior.

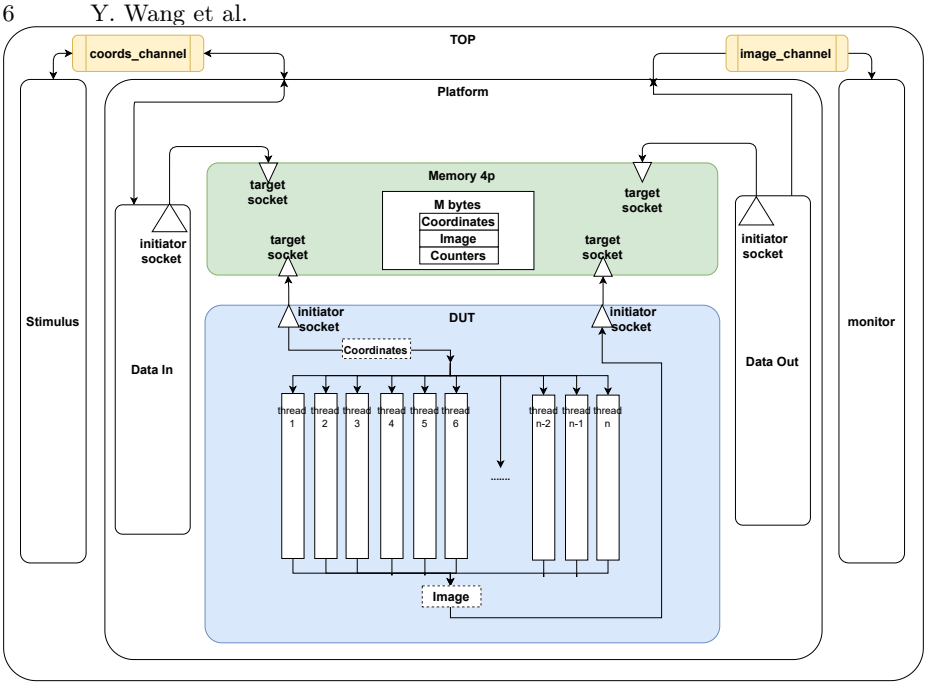

Fig. 4. Schematic of Theoretical Mandelbrot Set Visualization Model

Overview of Theoretical Model As shown in Figure 4, the highest level module that contains all submodules is named *TOP*. *TOP* contains module *Stimulus, Monitor, Platform*, and two communication *Channels*. Module *Platform* contains module *Data In, Data Out, four-port Memory*, and *DUT*.

Stimulus is in charge of creating the coordinates used for the Mandelbrot algorithm. The coordinates have a custom data structure with four floating point numbers that represent the top, left, right, and bottom bounds. Because each image requires a coordinate, the number of coordinates generated from stimulus equals the number of images. Each coordinate generated after the first one will be zoomed in based on a zoom factor. The zoom factor is user-defined and defaults to 0.7.

Platform contains module *Data In, Data Out*, and *DUT*. The module in charge of communication between *Stimulus* and *Platform* is a SystemC channel class. When a coordinate reaches *Platform*, it is received by the module *Data In*. *Data I*n runs an infinite loop of sending coordinates right after receiving coordinates from *Stimulus*. Similarly, *Data Out* runs an infinite loop of sending images to *Monitor*. The purpose of *Data In* and *Data Out* is to represent I/O units.

Module *DUT* wraps all the Mandelbrot parallel units, as shown in Figure 4. The number of parallel units n used is a preprocessor macro and can be power of 2 with a maximum of 256. Each parallel unit calculates $\lfloor \frac{heightofimage}{numberofparallelunits} \rfloor$ rows of pixels. If the height of the image is not divisible by n, the last parallel unit will compute the extra rows of pixels.

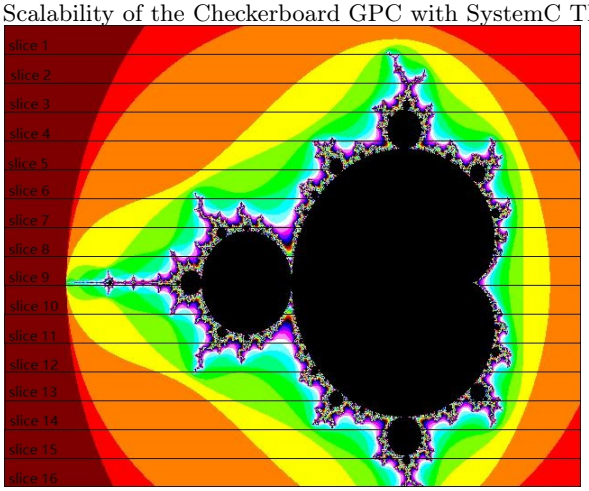

Fig. 5. Example of 640 * 512 Mandelbrot with 16 Parallel Units

All parallel units work on the same instance of image, as illustrated in Figure 5. An image consists of three 2D array of values to store a pixel's red, green and blue intensity. Inside each thread, the function $f_c(z_{n+1})) = z_n^2 + c$ is executed in a loop to determine the number of iterations each pixel takes to breach the threshold. After getting the iteration number for a pixel, that number is mapped to a colored pixel on the image. After calculating and displaying all assigned pixels, the thread pauses and waits for the next coordinate to start the next image. When the entire image is filled, DUT sends the image to memory and reads the next coordinate. The Monitor is in charge of stopping the program when it receives the expected number of images.

3.2 Mapping on the Checkerboard Model

Figure 6 shows our mapping of Mandelbrot slices on each *Core* in the Checkerboard model. Each image is divided into $GridWidth * GridHeight$ slices, and each *Core* in the mapped Checkerboard is in charge of 1 slice of the image. In this example, each black line in Figure 6 represents a flow of 4 slices of the image.

The simplified dataflow of a single column of Mandelbrot on Checkerboard 4x4 is shown on the right of Figure 6. A red block represents a *Core*, and a blue block represents a memory. Every *Core* uses the same functions: PopCoord(), PushCoord(), PopSlice(), and PushSlice(). PopCoords() takes the generated coordinates from the memory above that *Core*. If the *Core* is in the first row, then PopCoords() take the coordinate from *Off-Chip Memory*. Function PushCoords() pushes a coordinate to the local memory of that *Core*. PushSlice() function writes a finished image slice to a *Core*'s local *On-Chip Memory*, if *Core* is in the last row, then PushSlice() write to *Off-Chip Memory* at the bottom. PopSlice() reads a image slice from the memory of the Cell above.

When a *Core* receives a coordinate for Mandelbrot calculation, it forwards that coordinate to the next *Core*. That *Core* then starts the Mandelbrot calculation function. Once all the pixels in a slice are calculated, the *Core* pushes this

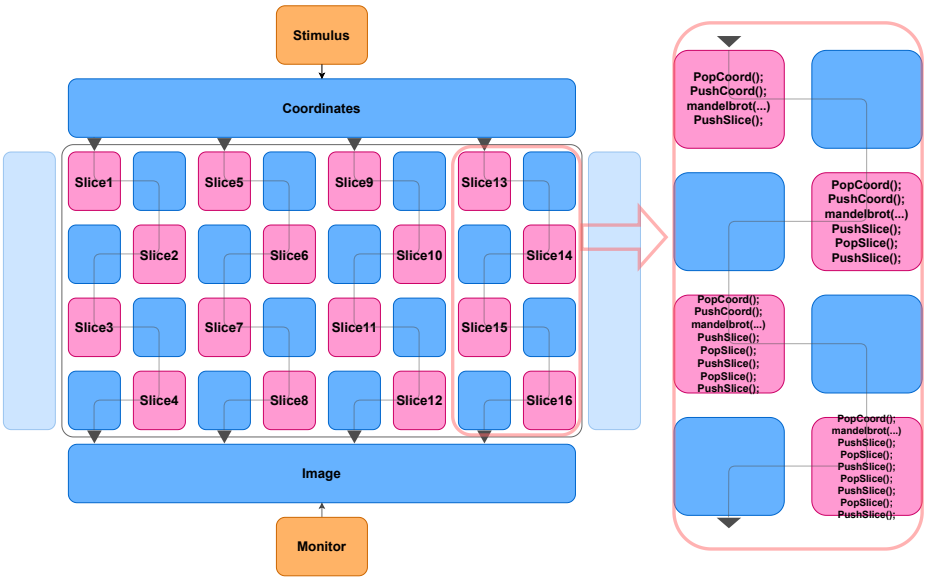

Fig. 6. Mapping of Mandelbrot slices on a Checkerboard 4x4 Model

slice to memory. For *Cores* that are not in the first row, the number of Pushing and Popping depends on the vertical location of that *Core*.

Module *Monitor* checks the bottom *Off-Chip Memory* for every write. It waits until every slice of the image is filled and then displays a message that an image has been completed. Once the number of images matches the specified number, *Monitor* terminates the simulation.

4 Experiments and Results

This section introduces our experimental setup and testing methodology for Mandelbrot Set Visualization on the theoretical model and on the Checkerboard model. All recorded results are in unit time since the actual timing delay for *On-Chip Memory*, *Off-Chip Memory*, *Multiplexers* and *Demultiplexers* is irrelevant[1] for our goal of showing scalability.

4.1 Experimental Setup

All experiments use the same parameters, the only difference is the number of parallel units. Experiments run for Mandelbrot Set Visualization use the following parameters: 576 Image Height, 640 Image Width, 4096 Max Iteration, 5

[1] Please note that the irrelevant timing delays do not include *contention* time when modules wait for access to shared bus resources. Contention is a very relevant criterion when comparing system architectures with shared resources and would be highly desirable in our comparison. However, measurement of contention was unfortunately not yet available in the described models at the time of the experiments reported in this section.

Images. All experiments are run on the same machine with fixed frequency of 3.4GHz. For repeatability we turn off all I/O functions and frequency scaling of the host. 1 computation unit time equals 1 iteration taken in the Mandelbrot Set calculation loop. 1 communication unit time equals 1 word read/write from/to the memory and 1 read/write request forwarded by the mux/demux.

4.2 Results and Evaluation for the Theoretical Model

We built and simulated the theoretical model in SystemC TLM-2.0. Here we report for an increasing number of parallel units (PU), the computation unit time (Comp UT), the communication unit time (Comm UT), the simulator run time (SRT), and improvement factor (IF). IF is calculated from $\frac{previousCompUT}{currentCompUT}$.

Table 1. Experimental Results for Mandelbrot Set Visualization Theoretical Model

PU	Comp UT	Comm UT	SRT (sec)	IF
1	4075078882	1658988	18.45	
2	2543059511	1658988	18.85	1.602431585
4	1471609768	1658988	19.06	1.728080070
8	764406229	1658988	19.49	1.925167159
16	389276159	1658988	19.83	1.963660531
32	196793623	1658988	19.89	1.978093360
64	98881005	1658988	20.24	1.990206542

Table 1 shows the simulation results for the theoretical model. For a growing number of parallel units, the calculation time decreases almost proportionally while the communication time remains constant due to the single memory bus. Taking into account that the slices require a different number of iterations (slices in the middle of the image are more expensive to calculate), both observations match the expectation for the otherwise perfectly scalable model. The simulator runtime increases slightly, which is also expected as the simulator needs to manage more threads and has increasing context switching activity. The values of Table 1 are also plotted in Figures 7 and 8 for comparison.

4.3 Results and Evaluation for the Checkerboard Model

Table 2 shows the simulation results for the Checkerboard model. The improvement factor in Table 2 is only calculated when number of PU matches Table 1. For a growing number of parallel units, the calculation time decreases proportionally while the communication time varies based on the layout. The observed calculation time matches our expectation for the same reason as in the theoretical model. Because the Checkerboard model features multiple Cells reading and writing to memories at the same time, the layout of the Checkerboard model reduces the communication time since the model assumes a contention-free *Off-Chip Memory*. For example, although the 2x2 layout has more PU, the 1x3 layout has one more column, so the 1x3 layout has less communication time than the 2x2 layout. The values of Table 2 are also plotted in Figures 7 and 8 for comparison.

Table 2. Experimental Results for Mandelbrot on Checkerboard Model

PU	Layout	Comp UT	Comm UT	SRT	IF
1	1x1	4075078982	1659029	18.61	
2	1x2	2543059611	829598	18.72	1.602431561
3	1x3	1799580598	553130	19.22	
4	2x2	1471609868	1106205	19.30	1.728080021
6	2x3	1001359981	553151	19.40	
8	2x4	748676317	553270	19.69	1.965615627
9	3x3	677227735	430285	19.87	
12	3x4	513788640	530309	19.96	
16	4x4	388649106	484295	20.26	1.926355433
20	5x4	304283538	430599	20.00	
24	6x4	260990624	461403	20.19	
28	7x4	218037155	423066	19.75	
32	8x4	196733139	415443	20.07	1.975514181
36	9x4	174894521	398790	20.19	
40	10x4	153292084	417555	19.88	
44	11x4	142358440	400279	20.23	
48	12x4	131401723	404199	20.52	
52	13x4	120450230	391782	20.17	
56	14x4	109542155	394545	20.34	
60	15x4	98877301	416078	19.75	
64	16x4	98877301	381393	20.36	1.989669388

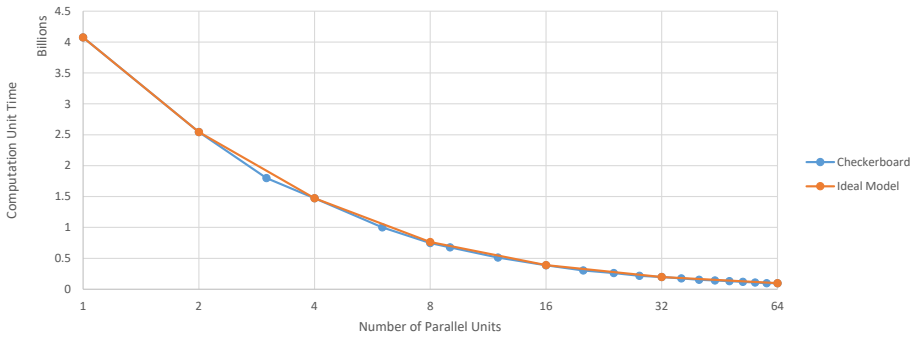

Fig. 7. Computation Unit Time vs Number of Parallel Units Comparison

4.4 Comparison

The improvement factors in Tables 1 and 2 are listed for all cases where the PU double. The values come close to the naively expected value 2.0, but do not fully reach this perfect score due to differences in slice complexity. Table 1 shows that the theoretical model is computationally scalable. Figure 7 shows that the Checkerboard model in this graph has an almost identical curve to the theoretical model. Therefore, the Checkerboard model is also computationally scalable.

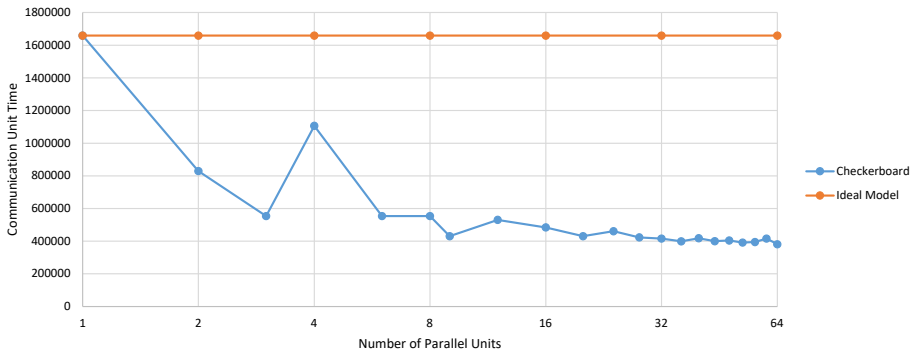

Fig. 8. Communication Unit Time vs Number of Parallel Units Comparison

The theoretical model has a single data bus and writes the entire image after all parallel units are done working. Therefore the communication unit time remains constant for different numbers of parallel units. Figure 8 shows that the Checkerboard model (except 1x1 layout) always has less communication time than the theoretical model. Because the Checkerboard model allows multiple Core-Memory-Core communication at the same time, the Checkerboard model has better communication scalability than the theoretical model.

5 Conclusion

In this work [19], we deliver a simulatable SystemC Checkerboard Grid of Processing Cell model and show that scalable software can be mapped onto this model. Our results show that the Checkerboard model has better communication performance and almost identical computation performance to the theoretical model, confirming that the Checkerboard GPC scales well. The Checkerboard model, while still a high-level software model, is a good starting point for a more complex and accurate SystemC model. This Checkerboard project is also proven to be a stable from other mapping projects [5] [7]. The Checkerboard project also serves as a stable and flexible platform for more software simulations and enables further explorations of the true scalability of the Checkerboard GPC architecture. In future work, we aim to automate the mapping of applications to GPC platforms so that they can be programmed similar as regular shared memory architectures.

References

1. The Tile Processor™ architecture: Embedded multicore for networking and digital multimedia. In: 2007 IEEE Hot Chips 19 Symposium (HCS). pp. 1–12 (2007). https://doi.org/10.1109/HOTCHIPS.2007.7482495
2. IEEE Standard for Standard SystemC Language Reference Manual. IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005) pp. 1–638 (2012). https://doi.org/10.1109/IEEESTD.2012.6134619
3. Bohnenstiehl, B., Stillmaker, A., Pimentel, J.J., Andreas, T., Liu, B., Tran, A.T., Adeagbo, E., Baas, B.M.: Kilocore: A 32-nm 1000-processor computational array.

IEEE Journal of Solid-State Circuits **52**(4), 891–902 (2017). https://doi.org/10.1109/JSSC.2016.2638459
4. Charlie Demerjian: A look at the 100-core Tilera Gx. https://www.semiaccurate.com/2009/10/29/look-100-core-tilera-gx/ (10 2009), [Online; accessed 30-May-2022]
5. Daroui, A.: A loosely timed TLM 2.0 Model of a JEPG encoder on a Checkerboard GPC. Tech. Rep. CECS TR 22-04, University of California, Irvine (October 2022)
6. Dömer, R.: A Grid of Processing Cells (GPC) with Local Memories. Tech. Rep. Technical Report 22-01, UCI, Center for Embedded and Cyber-Physical Systems (April 2022)
7. Govindasamy, V.B.: A tlm 2.0 model of a png encoder on a checkerboard gpc. Tech. Rep. CECS TR 22-02, University of California, Irvine (September 2022)
8. Held, J.: "single-chip cloud computer", an ia tera-scale research processor. In: Guarracino, M.R., Vivien, F., Träff, J.L., Cannatoro, M., Danelutto, M., Hast, A., Perla, F., Knüpfer, A., Di Martino, B., Alexander, M. (eds.) Euro-Par 2010 Parallel Processing Workshops. pp. 85–85. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
9. Hennessy, J.L., Patterson, D.A.: Computer Architecture: A Quantitative Approach. Morgan Kaufmann, Amsterdam, 5 edn. (2012)
10. Intel Labs: Introducing the Single-chip Cloud Computer. https://simplecore.intel.com/newsroom-en-eu/wp-content/uploads/sites/13/2010/05/Intel_SCC_whitepaper_4302010.pdf, [Online; accessed 30-May-2022]
11. von Neumann, J.: First draft of a report on the EDVAC. Tech. rep., University of Pennsylvania (June 1945)
12. Patterson, D.A., Hennessy, J.L.: Computer Organization and Design, Revised Fourth Edition, Fourth Edition: The Hardware/Software Interface. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edn. (2011)
13. Peh, L.S., Keckler, S.W., Vangal, S.: On-Chip Networks for Multicore Systems, pp. 35–71. Springer US, Boston, MA (2009). https://doi.org/10.1007/978-1-4419-0263-4_2, https://doi.org/10.1007/978-1-4419-0263-4_2
14. Robert Brooks and Peter Matelski: The dynamics of 2-generator subgroups of psl(2,c). Irwin Kra (ed.) (1978)
15. Taylor, M.B., Kim, J.S., Miller, J.E., Ghodrat, F., Greenwald, B., Johnson, P.R., Lee, W., Ma, A., Shnidman, N.R., Wentzlaff, D., Frank, M.I., Amarasinghe, S.P., Agarwal, A.: The raw processor: A composeable 32-bit fabric for embedded and general purpose computing (2001)
16. Tilera: Manycore without Boundaries: TILE64 Processor. http://www.tilera.com/products/processors/TILE64, [Online; accessed 30-May-2022]
17. Tilera: Manycore without Boundaries: TILEPro64 Processor. http://www.tilera.com/products/processors/TILEPRO64, [Online; accessed 30-May-2022]
18. Vangal, S., Howard, J., Ruhl, G., Dighe, S., Wilson, H., Tschanz, J., Finan, D., Iyer, P., Singh, A., Jacob, T., Jain, S., Venkataraman, S., Hoskote, Y., Borkar, N.: An 80-tile 1.28tflops network-on-chip in 65nm cmos. In: 2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers. pp. 98–589 (2007). https://doi.org/10.1109/ISSCC.2007.373606
19. Wang, Y.: A Scalable SystemC Model of a Checkerboard Grid of Processing Cells. Tech. Rep. CECS TR 22-03, University of California, Irvine (October 2022)
20. Wentzlaff, D., Griffin, P., Hoffmann, H., Bao, L., Edwards, B., Ramey, C., Mattina, M., Miao, C.C., Brown III, J.F., Agarwal, A.: On-chip interconnection architecture of the tile processor. IEEE Micro **27**(5), 15–31 (2007). https://doi.org/10.1109/MM.2007.4378780