Center for Embedded and Cyber-Physical Systems
University of California, Irvine

A Grid of Processing Cells (GPC) with Local Memories

Rainer Dömer

Technical Report CECS-22-01
April 18, 2022

Center for Embedded and Cyber-Physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
(949) 824-8919

doemer@cecs.uci.edu
http://www.cecs.uci.edu

A Grid of Processing Cells (GPC) with Local Memories

Rainer Dömer

Technical Report CECS-22-01
April 18, 2022

Center for Embedded and Cyber-Physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
(949) 824-8919

doemer@cecs.uci.edu
http://www.cecs.uci.edu

Abstract

In this position paper, we briefly review traditional single-, multi-, and many-core computer architectures which suffer from the well-known memory bottleneck between the processor(s) and the single shared main memory. After establishing the fact that the memory bottleneck delays many-core processors for thousands of cycles, we then propose an alternative Grid of Processing Cells (GPC) *structure of many cores with local memories that are arranged in a scalable 2-dimensional array with only local interconnect. We then specify three variants of the proposed GPC with hierarchical, "checkerboard", and 3D connectivity between the cells of processors with local memories. These GPC variants can serve as a starting point for studying, modeling, simulating, and exploring the vision of future computing platforms without a shared memory bottleneck.*

Contents

List of Figures

List of Tables

A Grid of Processing Cells (GPC) with Local Memories

Rainer Dömer
Center for Embedded and Cyber-Physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
doemer@cecs.uci.edu
http://www.cecs.uci.edu

Abstract

In this position paper, we briefly review traditional single-, multi-, and many-core computer architectures which suffer from the well-known memory bottleneck between the processor(s) and the single shared main memory. After establishing the fact that the memory bottleneck delays many-core processors for thousands of cycles, we then propose an alternative Grid of Processing Cells (GPC) *structure of many cores with local memories that are arranged in a scalable 2-dimensional array with only local interconnect. We then specify three variants of the proposed GPC with hierarchical, "checkerboard", and 3D connectivity between the cells of processors with local memories. These GPC variants can serve as a starting point for studying, modeling, simulating, and exploring the vision of future computing platforms without a shared memory bottleneck.*

1 Introduction and Motivation

Ordinary and embedded computer systems have a profound impact on our everyday life and society. With applications ranging from video-enabled mobile devices to real-time automotive applications and reliable medical devices, we interact with and depend on computer systems on a daily basis.

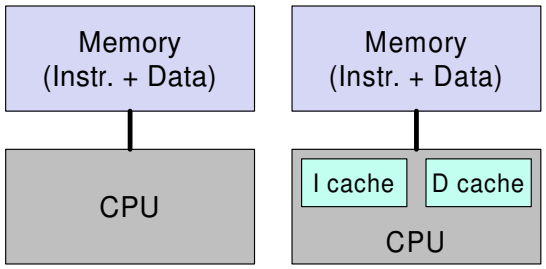
General-purpose and in particular special-purpose embedded computer systems are designed with multiple conflicting goals and requirements, including high speed, complex functionality, strict timing, low power, high reliability, as well as efficient production and maintenance. At the same time, customers constantly demand lower prices and a shorter time-to-market for the multitude of digital computer systems around us.

Among the many constraints and limitations of today's computing devices, one of the biggest open problems is the *memory bottleneck*. Observed already in the first single-processor architectures, the single lane through which data and instruction streams are transferred to and from the main memory impedes the traffic flow and often results in heavy congestion, limiting the processing speed severly. This problem is only multiplied in today's shared-memory multi- and many-core architectures and, despite sophisticated multi-level cache hierarchies, remains as a *grand challenge* that stands in the way of efficient parallel processing.

While Moore's law is coming to an end due to the physical limits on further increasing clock frequencies, the demand for parallel computing will only increase with the advance of big data and deep learning applications. Realizing the required computing performance with efficient processor architectures requires novel parallel platforms that do not suffer from a central memory bottleneck (and ideally do not need any costly cache hierarchies).

In this report[1] we briefly review the main memory bottleneck in current single-, multi-, and many-

[1] This report is a *position paper* (or *white paper*) that formally documents an idea that the author has shared [1] and discussed [2] with the students in his laboratory since March 2021.



(a) von-Neumann (b) modified Harvard

Figure 1: Classic computer architectures.

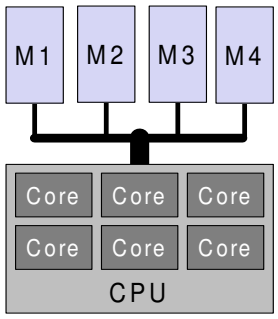


Figure 2: Multi-core architecture with multi-channel memory.

core computer architectures (Section 2), establish the severe delays it causes (Section 2.1), and then propose an alternative *Grid of Processing Cells (GPC)* approach (Section 3) where pairs of cores and local memories are arranged in a scalable 2-dimensional array. After a generic GPC is defined in Section 3.1, three GPC variants are specified in detail with local interconnect only, namely a hierarchical GPC (Section 3.2), a "checkerboard" GPC (Section 3.3), and a 3D GPC (Section 3.4). We finally conclude this report in Section 4 with possible future work where the proposed GPC variants can serve as a starting point for studying, modeling, simulating, and exploring future computing platforms with local memories.

2 The Main Memory Bottleneck

The traffic congestion of data and instruction streams through a single memory bus has already been observed in the classic von-Neumann computer architecture [3], as depicted in Figure 1(a). While the original Harvard architecture [4] featured separate memories for instructions and data, its modern implementation, the so-called modified Harvard architecture [5], also uses only a single shared bus to the main memory (despite its internally separated instruction and data caches), as shown in Figure 1(b).

While today's computers are typically organized as symmetric multiprocessors (SMPs) [6] and feature multiple (or many) processing cores on a single chip, they still use a single shared memory connected through a bottleneck bus interface. Even in modern computer architectures with multi-channel memory, as depicted in Figure 2, the processor cores share a single bus interface to the memory controller[2].

In order to compensate for the traffic congestion created by the shared memory bottleneck, usually a *memory hierarchy* is deployed with multiple levels of *cache*[3] memories between the processing cores and the main memory. A typical situation is shown in Figure 3 where three levels of caches are used. Each core has a private L1 cache, shares a L2 cache with a neighbor, and finally shares a L3 cache with all other cores on the same chip.

Note that the memory hierarchy is ordered by access speed. The closer the cache is to the core, the faster it must deliver data for cache hits. As a reference, Table 1 lists typical access times and storage sizes for different memories in the hierarchy [4].

The on-chip cache memories are typically implemented as static random access memory (SRAM), whereas the shared main memory typically consists of dynamic random access memory (DRAM) and is placed off-chip due to the differences in manufacturing technology. Here it is important to note that the caches occupy a significant amount of area on the processor chip which is very expensive (despite being logically redundant).

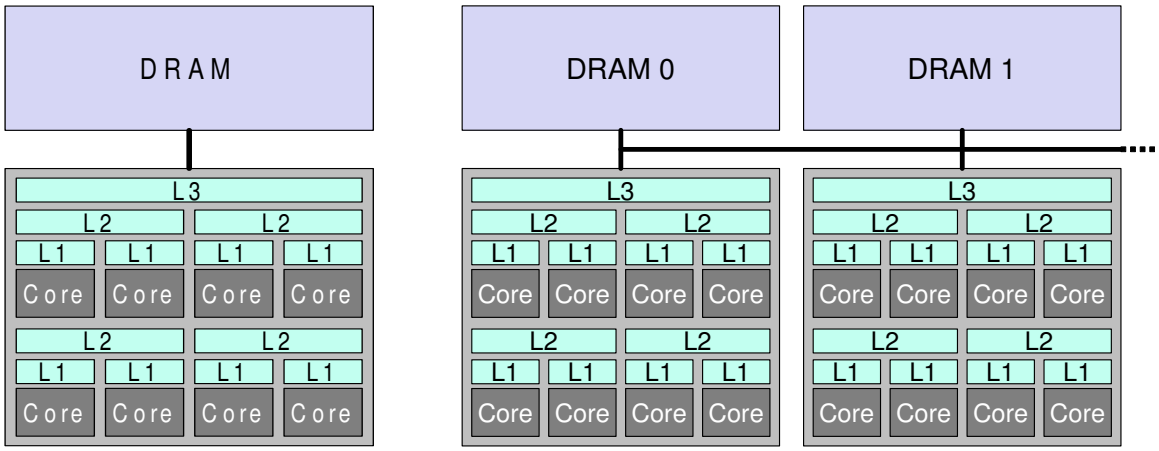
While in multi-core architectures typically all cores have the same uniform access to the shared memory, as depicted in Figure 3(a), special non-uniform memory access (NUMA) architectures exist as well, as il-

[2]Here multiple parallel communication channels exist only between the main memory and the memory controller, not between the controller and the cores.

[3]A cache is a small fast memory that can exploit data locality by keeping most recently used data available on-chip and thus reduce the traffic to the main memory.

Computer		Register	L1 Cache	L2 Cache	L3 Cache	Memory
Server	Size	1000 B	64 KB	256 KB	2–4 MB	4–16 GB
	Speed	300 ps	1 ns	3–10 ns	10–20 ns	50–100 ns
Mobile Device	Size	500 B	64 KB	256 KB		256–512 MB
	Speed	500 ps	2 ns	10–20 ns		50–100 ns

Table 1: Typical memory hierarchy with multiple levels of caches [4].



(a) Uniform memory access

(b) Non-uniform memory access (NUMA)

Figure 3: Multi-core architectures with hierarchical levels of caches.

lustrated in Figure 3(b). With NUMA, multiple memories (which cover different ranges in the shared address space) are available with different distances to the processors. As a result, each processor can access its closest memory quickly and such local accesses can occur in parallel. Accesses to non-local memories are possible too, but need to go through the interconnect (e.g. crossbar) and thus are significantly slower.

2.1 Intel Xeon Phi Example

As one real-world example, we quantify the main memory bottleneck for an industrial many-core processor, namely the Intel Many Integrated Core (MIC) architecture. Specifically, Figure 4 depicts the structure of the Xeon Phi coprocessor chip [7] which can be classified as a symmetric multiprocessor (SMP) with shared uniform memory access (UMA) [8].

The single die of the Xeon Phi 5120D chip integrates 60 processing cores, each based on the x86 instruction set architecture (ISA). The parallel processing cores communicate via a high performance bidirectional ring interconnect. Each core includes

Figure 4: Intel Xeon Phi coprocessor architecture with bidirectional ring interconnect [7].

a 32 KB L1 instruction and data cache, as well as a private 512 KB L2 cache and can fetch and decode instructions in-order from four hardware thread contexts. Thus there are in total 240 logical cores available for highly parallel processing.

We have measured the main memory access latency using a simple ping-pong algorithm for an idle situation (2 cores communicate, all other cores are idle)

and also for a busy situation (half of the cores are communicating, the other half are idle) [9]. Our statistical evaluation of the measurement results shows that the memory access latency for the Xeon Phi coprocessor reaches hundreds to thousands of cycles. Especially when multiple pairs of communication occur concurrently, the core-to-memory communication becomes very expensive and unpredictable.

Figure 5 plots the core-to-memory-to-core distances from core 0 to core $4n+1$ ($0 \leq n \leq 59$) on a busy coprocessor (green line), in comparison to the idle situation when only two cores communicate with each other (blue line). On an idle chip, the core-to-core latency is relatively stable between 1,000 and 1,500 cycles. The corresponding core-to-memory latency is more than 600 cycles on average. On the other hand, when half of the chip is busy and 120 cores compete for access to the memory, the core-to-core communication latency varies widely and grows up to 10,000 cycles for one round trip [9].

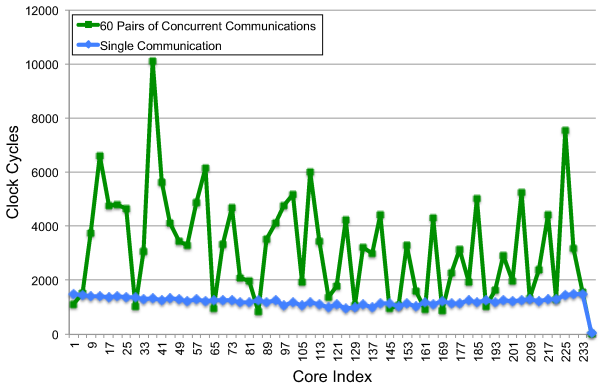


Figure 5: Intel Xeon Phi memory latencies measured from core 0 to memory to other cores at 1% (blue) and 50% (green) core utilization [9].

3 Grid of Processing Cells (GPC)

In order to avoid the traffic jam through the memory bottleneck which persists despite costly cache hierarchies, we propose a scalable multiprocessing approach where processing cores are paired with local memories. Placing many such pairs on the chip can ultimately eliminate the need for expensive caches. In other words, we envision many-core processors where on-chip caches are replaced with local memories[4].

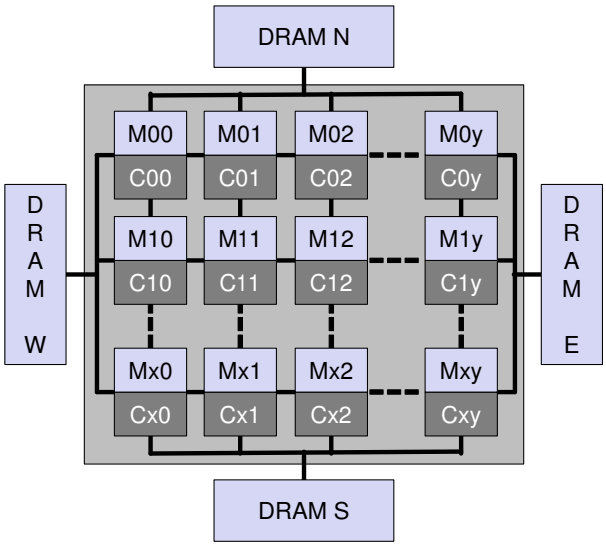


Figure 6: Proposed Grid of Processing Cells (GPC).

Similar to standard logic cells which paved the way to scalable digital circuits design, we propose a *Grid of Processing Cells (GPC)* structure where processor-memory pairs are arranged on-chip in a two-dimensional array with local interconnect. Using Flynn's taxonomy [12], such a GPC processor classifies as a multiple instruction streams, multiple data streams (MIMD) architecture.

3.1 Generic GPC Architecture

In a generic GPC as shown in Figure 6, each cell consists of a fully equipped general-purpose processing core Cij with its own local memory Mij of substantial size and high speed (SRAM). Pairs of processing cores and local memories are placed on the chip in a two-dimensional array of x by y cells and can be identified by row and column indices i and j, respectively.

In addition to its own local memory, each processing core also has access to the local memories of some of its closest neighbors. Subject to the specific GPC configuration chosen (three variants are described below), the access latency to the reachable memories will generally vary (NUMA). A core's own

[4] In embedded systems design, a similar approach is the use of scratchpad memory (SPM) for embedded processors [10, 11].

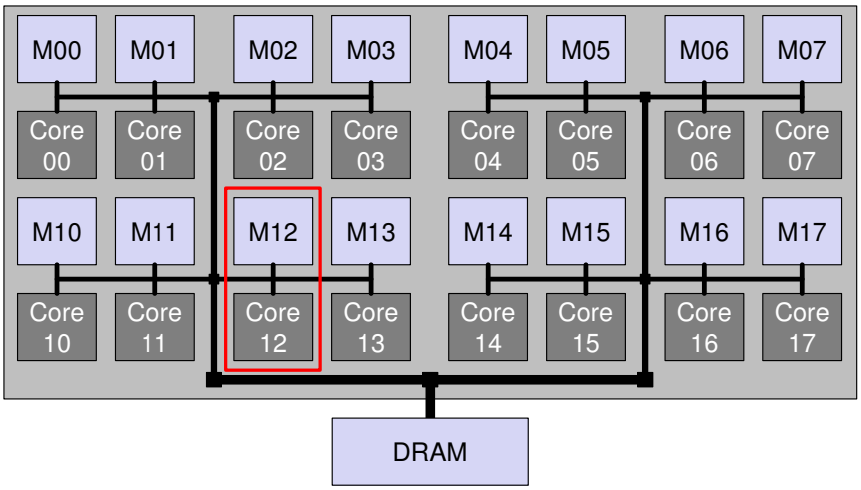


Figure 7: Hierarchical GPC architecture with 16 cells of processing cores with local memory.

local memory has the highest priority, followed by the neighbors' memories. The cores at the edges of the chip also have access to slower off-chip memory (large DRAM and/or memory-mapped I/O units).

While all GPCs are expected to follow a regular overall structure, the grid size (number of rows and columns), the cell parameters (e.g. homogeneous or heterogeneous core types and local memory size), and the connectivity among neighbor cells can be configured differently, for example, to match the desired application characteristics. The following sections specify three scalable GPC configurations that vary in the number of memories a processing core can access.

3.2 Hierarchical GPC Architecture

The on-chip connectivity between the processing cells of a GPC can be arranged hierarchically, for example, using an H-pattern. Figure 7 shows a *hierarchical GPC* architecture with 16 processing cells (PCs). Each PC is a pair of a processing core and a local memory, as indicated for cell 12 by the red border around Core12 and M12. Pairs of neighboring PCs are connected by a bus, and so are pairs of PC pairs, and so on. At the top of the interconnect hierarchy a large off-chip memory (DRAM) is available as well. Note that this hierarchical architecture technically allows every core to access every memory via the bus-based interconnect[5], but the access is faster the closer the memory is located to the processing core (NUMA).

3.3 "Checkerboard" GPC Architecture

The GPC on-chip interconnect can also be established as a regular 2D mesh of alternating processing cores and local memories. Figure 8 shows the so-called *"checkerboard" GPC* architecture in a 4-by-4 cell configuration. Due to the alternating placement pattern, we distinguish between L-type (green border) and R-type (red border) processing cells (PCs). Each PC again contains a processing core with local memory, but the core is located either at the left (L-type) or at the right side (R-type). In other words, odd row numbers are composed of R-type cells, even rows are L-type cells.

The 2D mesh interconnect allows each processing core to directly access its own and the memories of three of its neighbor cells, enabling extremely fast data transfer among neighboring processing cores. In Figure 8 for instance, Core11 owns its local memory M11, but also has immediate access to the neighbor memories M01, M12, and M21. At the chip edges,

[5]The access to every memory from every core in the proposed hierarchical GPC is in contrast to the "checkerboard" and 3D GPC architectures where access to memories outside of a cells immediate neighborhood can only be established indirectly with the explicit assistance of other cores' forwarding the data.

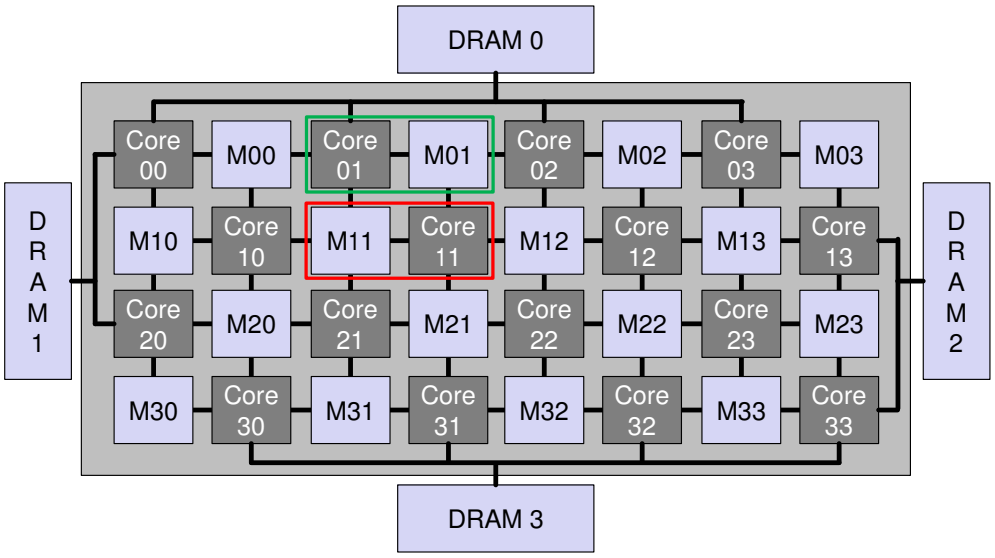


Figure 8: 4-by-4 "checkerboard" GPC with L-type (green) and R-type (red) processing cells.

we again connect additional large off-chip memories (DRAM) and/or memory-mapped I/O units.

On a "checkerboard" GPC in general, every processing unit $C^R_{i,j}$ in a R-type PC has direct access to its own memory $M_{i,j}$, as well as to its neighbor memories $M_{i-1,j}$, $M_{i,j+1}$, and $M_{i+1,j}$, whereas a core $C^L_{i,j}$ in a L-type PC has direct access to its own memory $M_{i,j}$, as well as to its neighbor memories $M_{i-1,j}$, $M_{i,j-1}$, and $M_{i+1,j}$. Whenever any index becomes negative or larger than the corresponding GPC chip dimension, the off-chip memories are accessed.

The "checkerboard" GPC connectivity can be established by priority-based multiplexing interconnect, as depicted for an R-type[6] PC in Figure 9. Shown by use of SystemC TLM-2.0 [13] initiator and target sockets, the PC Core is connected to its local memory via a de-multiplexer and a multiplexer (with highest priority). Surrounding neighbor memories are connected to the blue target sockets to the North, East, and South (with lower priority). Vice versa, surrounding neighbor cores can access the memory M via the gray initiator sockets from the North, West, and South (also with lower priority).

We note that the multiplexer-based interconnect in the "checkerboard" GPC goes hand in hand with the

[6]The corresponding L-type PC is a mirror image of the R-type figure.

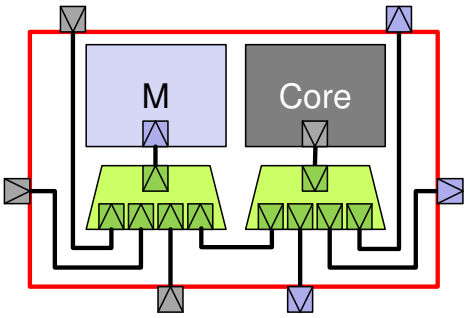


Figure 9: R-type processing cell (PC) in the "checkerboard" GPC with SystemC socket connectivity.

mapping of physical addresses assigned to the memories on- and off-chip. For example, a 32-bit address space can reserve 1 bit for on- vs. off-chip memory addresses, as well as 2 bits each for the on-chip memory indices of a 4-by-4 GPC. This way, the de-multiplexing logic needed in each cell can be of minimum complexity [2].

3.4 3D GPC Architecture

While the "checkerboard" GPC specified in Section 3.3 allows each processing core direct access to four memories (in North, East, South, and West direction), it can be easily extended with access to another local memory that may be placed conceptually (or

physically, if fabrication technology allows it[7]) atop (or below) each core. Such a *3D GPC* architecture extends the number of accessible memories for each processing core to five (namely the own local memory in addition to the neighbors' memories in North, East, South, and West direction) and no distinction between L-type and R-type PCs is necessary any more. Consequently, the multiplexing interconnect for the 3D GPC can be established with a trivial extension of the "checkerboard" scheme (Figure 9) by adding one extra port at the multiplexer and de-multiplexer.

4 Conclusion and Future Work

In this position paper, we have reviewed the main memory bottleneck in computers which causes severe performance degradation and, despite decades of research in computer architecture design, still persists with today's multi- and many-core processors. In order to address this grand challenge, we propose an alternative computer organization called Grid of Processing Cells (GPC) where processing cores are paired with local memories and are placed on the same chip in a scalable 2-dimensional array. In addition to a generic GPC, we specify three GPC variants in more detail, namely a hierarchical, a "checkerboard", and a 3D GPC.

While the proposed GPC approach jeopardizes the notion of a globally shared memory space and thus poses significant challenges to established software design and programming practices[8], it promises to largely remove the need for expensive multi-level caches and corresponding waste of precious chip area and high power consumption. Moreover, the GPC organization is scalable and thus can serve as a fabric for long-term growth in computing performance and truly parallel processing.

We are optimistic that the specified architecture variants can serve as valuable starting points for fruitful research to further study, model, simulate, evaluate, refine, and extend the proposed GPC as a promising future computing platform.

[7] The physical challenges to 3D chip manufacturing due to limitations in process and fabrication technology are outside the scope of this work.

[8] Such software design challenges are a promise of exciting future work.

References

[1] Rainer Dömer. Grid of processing cells (GPC). Personal communication, March 5 2021.

[2] Arya Daroui, Vivek Govindasamy, Yutong Wang, and Rainer Dömer. Modeling of a 4-by-4 checkerboard GPC in SystemC. Personal communication, October 7 2021.

[3] John von Neumann. First draft of a report on the EDVAC. Technical report, University of Pennsylvania, June 1945.

[4] John L. Hennessy and David A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2011.

[5] Wikipedia contributors. Modified harvard architecture — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Modified_Harvard_architecture&oldid=1070083742, 2022. [Online; accessed 13-April-2022].

[6] David A. Patterson and John L. Hennessy. *Computer Organization and Design - The Hardware / Software Interface (Revised 4th Edition)*. The Morgan Kaufmann Series in Computer Architecture and Design. Academic Press, 2012.

[7] Intel Corporation. Intel(R) Xeon Phi Coprocessor. *Datasheet, Reference Number: 328209-001EN*, November 2012.

[8] Intel Corporation. Intel(R) Xeon Phi Coprocessor System Software Developers Guide. *Report, SKU# 328207-001EN*, November 2012.

[9] Guantao Liu, Tim Schmidt, Ajit Dingankar, Desmond Kirkpatrick, and Rainer Dömer. Optimizing Thread-to-Core Mapping on Manycore Platforms with Distributed Tag Directories. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, January 2015.

[10] Preeti R. Panda, Nikil D. Dutt, and Alexandru Nicolau. Efficient utilization of scratch-pad memory in embedded processor applications. In *Proceedings of the European Design Automation Conference (Euro-DAC)*, pages 7–11, 1997.

[11] Rajeshwari Banakar, Stefan Steinke, Bo sik Lee, M. Balakrishnan, and Peter Marwedel. Scratchpad memory: A design alternative for cache on-chip memory in embedded systems. In *Proceedings of the International Symposium on Hardware-Software Codesign (CODES)*, pages 73–78. ACM, 2002.

[12] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9):948–960, 1972.

[13] IEEE Computer Society. *IEEE Standard 1666-2011 for Standard SystemC Language Reference Manual*. IEEE, New York, USA, 2011.