TYPEONEBIS

A SWARM TECHNOLOGY RISC-V CPU WITH TRILLIONS OF CORES AND HARDWARE THREADS: EMPLOYING SELF-ORGANIZATION TO REDUCE THREAD SWARM ENTROPY

White Paper

TABLE OF CONTENTS

Abstract

For decades CPUs have been confined to a limited number of cores. TypeOneBIS has developed a multi-trillion core RISC-V CPU that breaks these confines, applying swarm technology to self-organize and reduce entropy in the multi-trillion hardware thread swarm for smooth execution. There is no theoretical or practical limit to the number of cores a swarm-CPU can possess. Notwithstanding the ultra-futuristic technologies employed, it is easy to program the CPU, as it implements transparent swarm collective instructions, making it simple to control the multi-trillion thread swarm flow. The swarm-CPU is manufacturable using existing ASIC tools and fabrication processes. Expect in 2-4 years, multi-million core mobile phones, multi-billion core laptops and multi-trillion core workstations.

Introduction

In current state of the art general purpose CPUs, the number of cores is restricted as described by Amdahl's and Gustafson's Law. As of February 2024, the general purpose CPU with the largest number of cores is the AMD EPYC, with 128 cores and 256 hardware threads. Applying AI and simulation to real world scenarios, such as TypeOneBIS Biogenic Replicators and Biovisity Regulators, requires several million CPU nodes connected together via switches to form a large computer network infrastructure. The network introduces bottlenecks as the infrastructure scales (see figure 1). A collective operations library is often used to facilitate programming software to execute across this multi-CPU infrastructure.

In a multi-CPU process, individual CPU nodes compute partial results within an iteration frame, and use collective operations to aggregate the complete result. Subsequently, the result is shared among CPU nodes for use in computing the next frame. This process generates massive data traffic that can overwhelm the network, degrading performance dramatically. A key performance indicator (KPI) that can be used to benchmark multi-CPU architectures is the sustained number of iterated frames/second (frame rate). Arguably, a serious deficit in general purpose CPU designs, is their inability to transparently handle in hardware, multi-core, multi-CPU collective operations. Programming collective operation libraries for high performance requires detailed library understanding, making their use difficult if not intractable for all but expert developers. While the collective operations library solution works, it may introduce further frame rate degradation as software is generally thousands of times slower than hardware. The frame rate KPI is a useful throughput metric because the overall performance of a system is often limited by its slowest operation. Irrespective of the speed of the CPU nodes, an overwhelmed network will make node speeds irrelevant.

TypeOneBIS overcomes Amdahl's and Gustafson's Law using a more technologically advanced CPU architecture. The resulting general purpose, self-organizing, RISC-V CPU based on swarm technology, can consist of up to trillions of cores and trillions of hardware threads (harts). Swarms of trillions of harts self-organize, reducing entropy, which is key for smooth, coherent and collective execution. Swarm-CPU cores implement collective operation oriented instructions for managing hart swarms, making the CPU highly programmable. Hardware and software collective operations handling is depicted in figure 6 and 7. Due to the CPU design, the operating system can assign dedicated CPU cores to a hart-group tenant. This is important in a cloud computing environment where the CPU may be shared by multiple unrelated tenants, some with "egregious" behavior. The primary focus of this swarm-CPU series, is the achievable instructions per second (IPS) throughput using parallel execution.

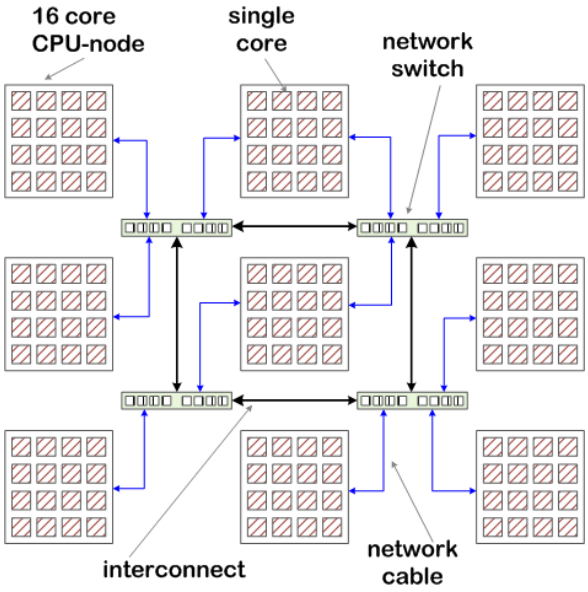


Figure .1

Current state of the art computing infrastructure consisting of 16-core CPU nodes, interconnected via switches. The number of CPU nodes can number in the millions (not shown).

TYPEONEBIS SWARM-CPU: SCALABLE TO UNLIMITED NUMBER OF CORES AND THREADS

Swarm-CPUs have a virtual memory page size of 4096 bytes. The max instruction per second (IPS) rate of cores is 1 instruction every 10 clock cycles (0.1 IPS). Figure 2-5 shows swarm-CPU specifications of various scales using a 5nm process.

MOBILE PHONE SWARM CPU

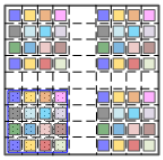


Figure .2 – **Mobile phone** swarm-CPU

- Frequency = 2.0GHz
- Chip size = 625mm^2
- Clusters = 10,241
- MMUs = 20,482
- Cache size = 83.9 MB
- Cores = 174,097
- Harts = 696,388
- Max Memory *(ram)* = 87.9 TB
- **IPS = 30.7 trillion**
- Cache speed = 131.1 TB/sec
- Memory speed = 8.2 TB/sec *

** Maximum possible bandwidth -- actual bandwidth depends on number of DRAM banks*

The mobile phone swarm-CPU has 20,482 MMUs, with up to 5,243,392 virtual pages resident in memory, mapping 21.47GBs of virtual memory into main memory. This makes it possible for a mobile phone to execute programs directly from a NAND flash hard drive obviating the need for RAM.

DESKTOP SWARM CPU

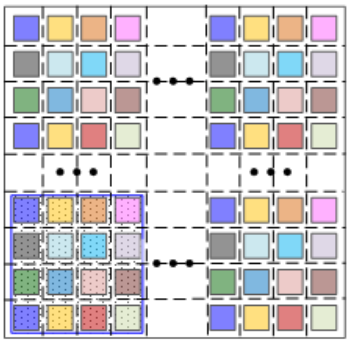
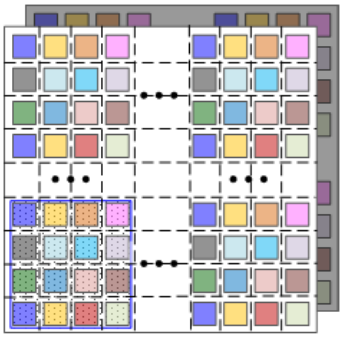


Figure .3 – **desktop** swarm-CPU

- Frequency = 4.0GHz
- Chip size = wafer-wide
- Clusters = 2,084,047
- Max Memory = 17.9 PB
- **IPS = 12,504.2 trillion**
- Cache speed = 53.3 PB/sec
- Memory speed = 166.7 TB/sec *
- MMUs = 4,168,094
- Cores = 35.42 million
- Harts = 141.71 million
- Cache size = 16.67GB

The desktop wafer-wide swarm-CPU has 4,168,094 MMUs, with up to 1,067,032,064 virtual pages resident in memory, mapping 4.37TBs of virtual memory into main memory.

EXA-SCALE SERVER SWARM CPU



"Game-changer" swarm-CPU costing only a few million USD

Figure .4 – Low price **Exa-scale** swarm-CPU

- Frequency = 4.0GHz
- Chip size = wafer-wide
- Wafer count = 320 3D stitched
- Clusters = 666.9 million
- MMUs = 1.33 billion
- Cores = 11.3 billion
- Harts = 45.3 billion
- Cache size = 5.46 TBs
- Max Memory = 5.72e18 Bytes
- **IPS = 4.0e18**
- Cache speed = 1.71e19 B/sec
- Memory speed = 533.3 PB/sec *
- **CPU size = .6m x .6m base, .8m height**

The exa-scale multi-wafer swarm-CPU has 1,333,790,080 MMUs, with up to 341.45 billion virtual pages resident in memory, mapping 1.39PBs of virtual memory into main memory. A future wafer-on-wafer manufacturing process will make this swarm-CPU small enough to be used as a workstation.

ZETA-SCALE SWARM CPU

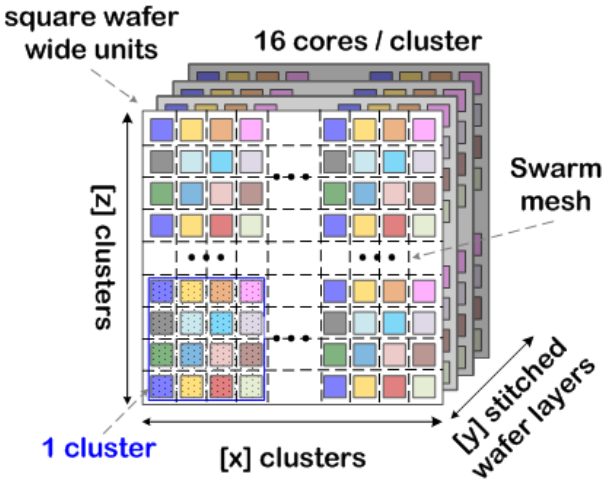


Figure .5 – **Zeta-Scale** swarm-CPU

- Frequency = 4.0GHz
- Chip size = wafer-wide
- Wafer count = 32k 3D stitched
- Clusters = 66.7 billion
- MMUs = 136.6 billion
- Cores = 1.16 trillion
- Harts = 4.64 trillion
- Cache size = 559.4 TBs
- Max Memory = 5.86e20 Bytes
- **IPS = 4.1e20**
- Cache speed = 1.75e21 B/s
- Memory speed = 5.46e19 B/s *
- **CPU size = 2.3m x 2.3m base, 5.2m**

The 546 TB cache size of the zeta-scale swarm-CPU is significant, since, for many real world applications the entire data set may fit in 546 TB. In such a case, the full data-set would reside in-cache and no page faults would occur, unleashing the full 4.1e20 IPS capability of the swarm-CPU. These CPUs are specifically designed to optimize for scenarios in which the full data set can fit in cache.

The zeta-scale multi-wafer swarm-CPU has 136,580,104,192 MMUs, with up to 34.96 trillion virtual pages resident in memory, mapping 143.21PBs of virtual memory into main memory.

ADAPTIVE CAPABILITY OF SWARM-CPUS

An important feature of the design is that the swarm mesh has an adaptive capability, to circumvent the expected manufacturing defects in a wafer-wide chip. On power reset, 'defective' clusters are pruned from the swarm-mesh, 'effectively' increasing chip yields to near 100%.

COLLECTIVE OPERATIONS

Definitions:

- **imm** – integer immediate value associated with an instruction.

Listed below are some **atomic** instructions added to the RISC-V set to support collective operations (see figure 6 and 7).

bv.wt rd, imm(rs1) :
Wait on semaphore, barrier capable instruction. This is a blocking operation as flow control is not returned to the hart until the memory value is tested and found to be equal to zero (==0).

bv.sg rs2, (rs1) :
Signal or initialize a barrier. Only one hart will ever successfully write its value to the barrier. This is a NO-OP for all other harts.

bv.tylk rd, (rs1) :
Try locking a critical section. This is a non-blocking operation as flow control is returned to the hart immediately whether or not the lock was obtained. If there is an "ABA" concern, the hart may use "LR/SC" instructions to ensure that during the operation, no other hart accessed the shared memory associated with the critical section lock.

bv.unlk rs2, (rs1) :
Unlock a critical section. This instruction can be used to serialize access to a critical resource shared by a hart-group.

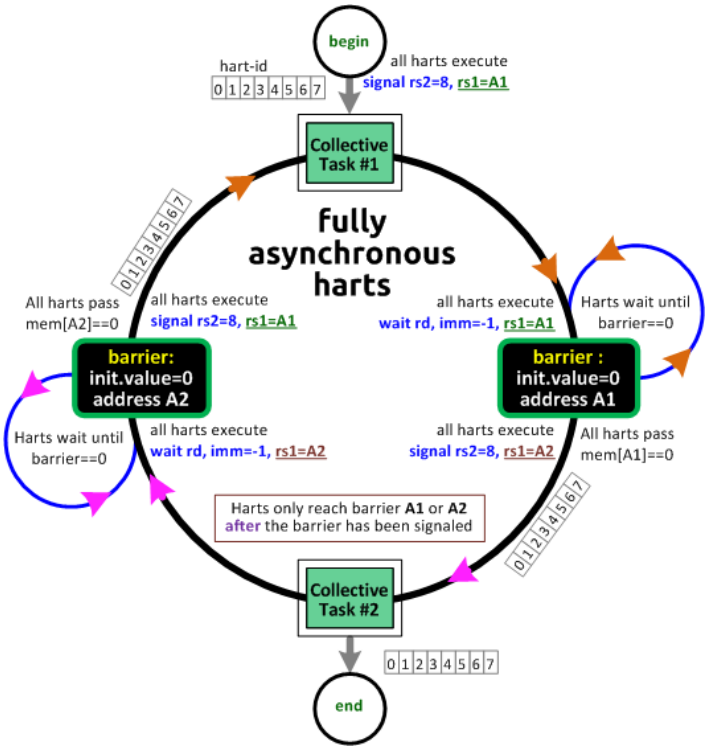


Figure .6 – **Barrier Collective operation**

Barriers **A1, A2** are initialized to zero. Eight (8) harts begin. All harts execute **signal (8, A1)**. One hart successfully sets A1==8, all other harts are ignored. Harts compute their **partial** result for collective task #1 and execute **wait (-1, A1)**, decrementing A1 on the ***first*** barrier encounter, and then wait until A1==0. When A1==0 all harts pass the barrier and immediately execute **signal (8, A2)**. One hart successfully sets A2==8, while all other harts are ignored. All harts contribute their partial result to the **complete** result at collective task #2 and then execute **wait (-1, A2)**, decrementing A2 on the ***first*** barrier encounter, and then wait until A2==0. When A2==0 all harts pass the barrier and immediately execute **signal (8, A1)**. One hart successfully sets A1==8, while all other harts are ignored. Harts use the **complete** result to compute the **next frame**. The procedure repeats until the process ends.

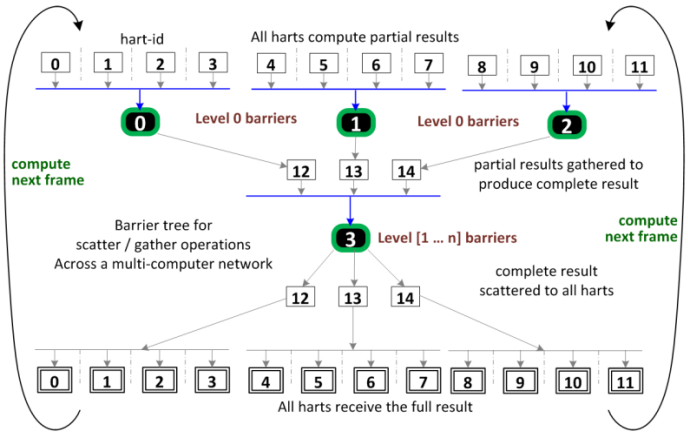


Figure .7 – **Multi-Computer barrier tree operation**

4 computers, C[0..3], C[4..7], C[8..11] and C[12..14] all have a barrier arrangement as in figure 6. C[0..3] create and forward its partial result over the network to hart[12]. C[4..7] forward to hart[13] and C[8..11] to hart[14]. Computers C[12..14] use barrier[3] to wait until all **partial** results have arrived before it proceeds to compute the final result. The final result is then sent back to all computers and used to compute the **next frame** after which the procedure repeats.

LINUX OPERATING SYSTEM

A modified version of Linux is the primary target operating system for swarm-CPUs since they are not bound by hart or MMU limitations. Usually CPUs have one or a few MMUs. To combat this scarcity, Linux manages PTEs using a page tree structure, quick-lists for fast access and statistics for each page. Additionally, until now, the number of harts in a CPU was 256 or less, so Linux uses context switching to share this limited resource among processes. The swarm-CPUs in figure 2 – 5, have millions to trillions of page table entries (PTE) and can have the same number of virtual pages resident in RAM, which may represent all of RAM. Linux requires simplification, since maintaining page trees, lists and statistics for millions to trillions of resident pages is intractable. The same goes for context switching millions to trillions of harts in any reasonable time.

CONCLUSION

The swarm-CPUs in figure 2 – 5 have conservative specifications. Swarm CPU cores do not yet use pipelining. Future pipelined cores will have at least 2 times the chip density and 2–4 times (or higher) the current IPS rate.

The exa-scale processing power and low price, of the swarm-CPU in figure 4, makes it affordable to many banks, telecoms, universities, laboratories, other companies and organizations. As a result it is expected that this swarm-CPU will be widely deployed. Furthermore, SRAM macros are much larger then swarm-CPU cores. When SRAM starts being manufactured like 3D-NAND, using stacked layers, a wafer-wide swarm-CPU will contain billions of cores.