



POLITECNICO
MILANO 1863

DEPARTMENT OF ELECTRONICS
INFORMATION AND BIOENGINEERING

September 1st, 2025

Fast Machine Learning for Science Conference 2025

SPARTA: High-Level Synthesis of Parallel Multi-Threaded Accelerators

09.1.2025 | Fabrizio Ferrandi, Serena Curzel, Michele Fiorito, Giovanni Gozzi, Vito Giovanni Castellana,
Marco Minutoli, Antonino Tumeo

Outlines

1. Introduction
2. SPARTA flow
3. SPARTA architecture
4. Results

Introduction

SPARTA is a methodology for synthesizing parallel hardware accelerators from high-level code:

- It starts from C/C++ programs annotated with OpenMP pragmas, enabling parallelism to be expressed at the software level.
- The output is a multi-core hardware accelerator capable of executing tasks concurrently.
- No hardware design knowledge is required — SPARTA abstracts away low-level details.
- It is fully integrated into the Bambu High-Level Synthesis (HLS) toolchain.

Introduction - OpenMP

Why OpenMP?

- It familiar to software developer
- Provides multiple work-sharing constructs and synchronization mechanisms
- It does not really matter: SPARTA only needs a way for the developer to describe parallelism.

SPARTA supports a subset of the OpenMP language:

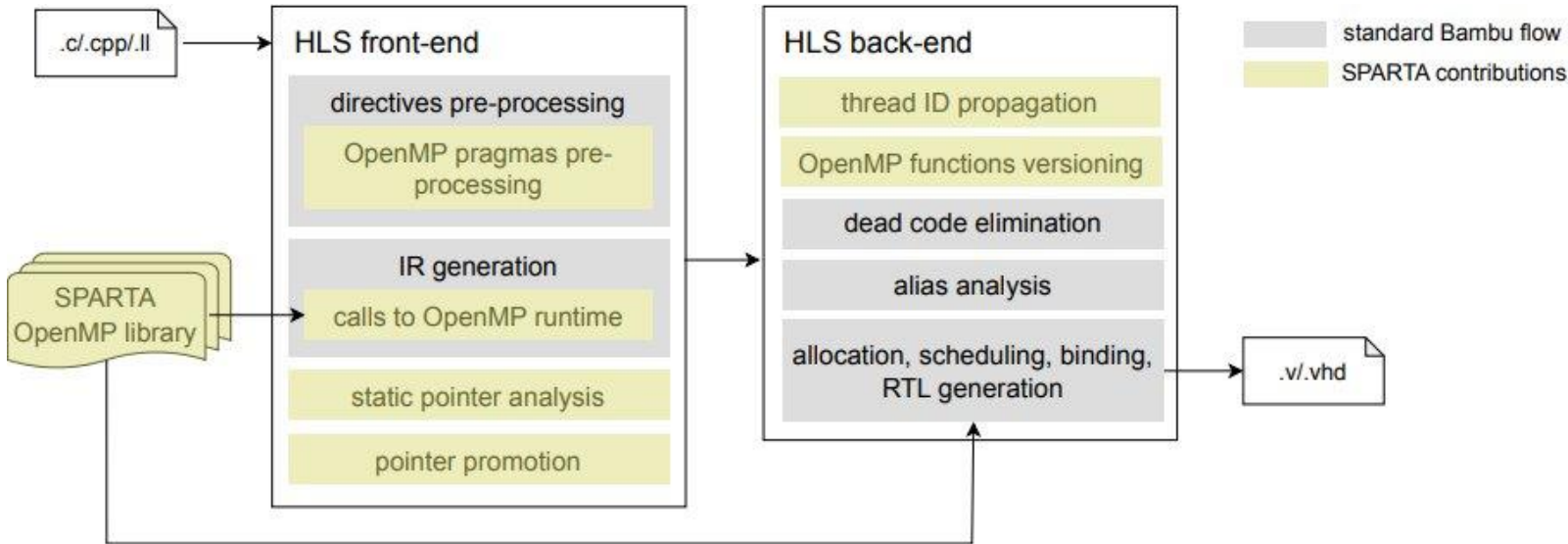
- Parallel for and sections
- Reduction, shared, private and firstprivate clause
- Critical operations
- Nested work-sharing constructs

OpenMP®

Example: Pi with a loop and a reduction

```
#include <omp.h>
static long num_steps = 100000;    double step;
void main ()
{
    int i;    double x, pi, sum = 0.0;
    step = 1.0/(double) num_steps;
    #pragma omp parallel
    {
        double x;
        #pragma omp for reduction(+:sum)
        for (i=0; i< num_steps; i++){
            x = (i+0.5)*step;
            sum = sum + 4.0/(1.0+x*x);
        }
    }
    pi = step * sum;
}
```

SPARTA flow



SPARTA modifies the execution flows of Bambu:

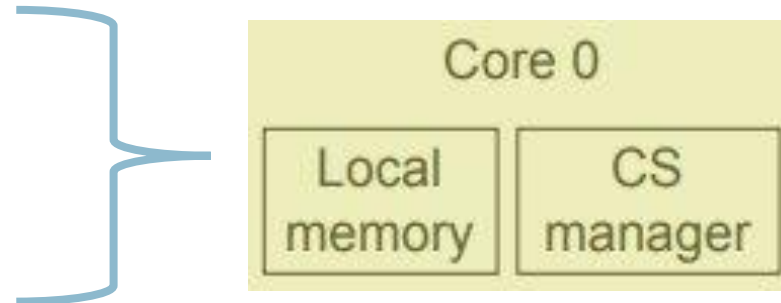
- It implements the operation of the OpenMP runtime with ad-hoc hardware components.
- Performs argument promotion to the arguments passed to the OpenMP runtime.
- It replicates the hardware creating multiple parallel cores.

SPARTA flow - example

```
void dot_product(int A[M], int B[M], int res[M]){
    int sum = 0;
    #pragma omp parallel num_threads(M) for
        for(i = 0; i < M; i++){
            prod(A[i], B[i], res[i]);
        }
}
```

```
define i32 @dot_product (...) {
    ...
    call void __kmpc_push_num_threads(M)
    call void __kmpc_fork_call(.omp_outlined., ...)
    ...
}
define void @.omp_outlined.(...) {
    %6 = call i32 @__kmp_bambu_tid_from_gtid(i32 %0)
    ...
    call void @prod(...)
    call void @__kmp_bambu_barrier_reached(i32 %6)
    call void @__kmp_bambu_wait_all_threads()
    ...
}
```

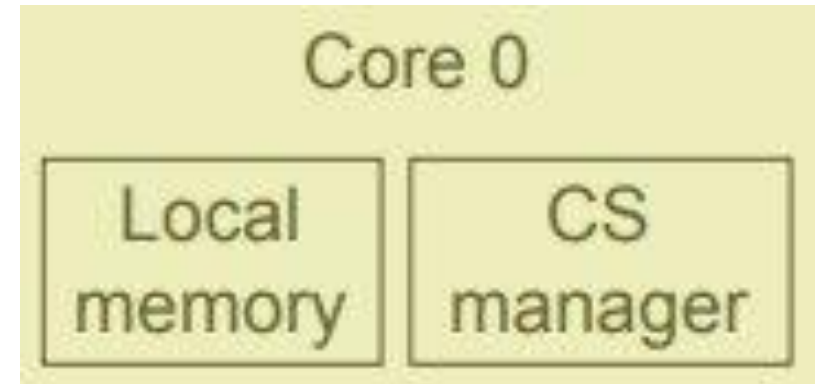
- The pragma in the annotated code are expanded into calls to the OpenMP runtime
- In software the fork call operation creates multiple threads: SPARTA maps each thread to a core (they can be shared).



SPARTA architecture

Cores are the base elements of the SPARTA architecture:

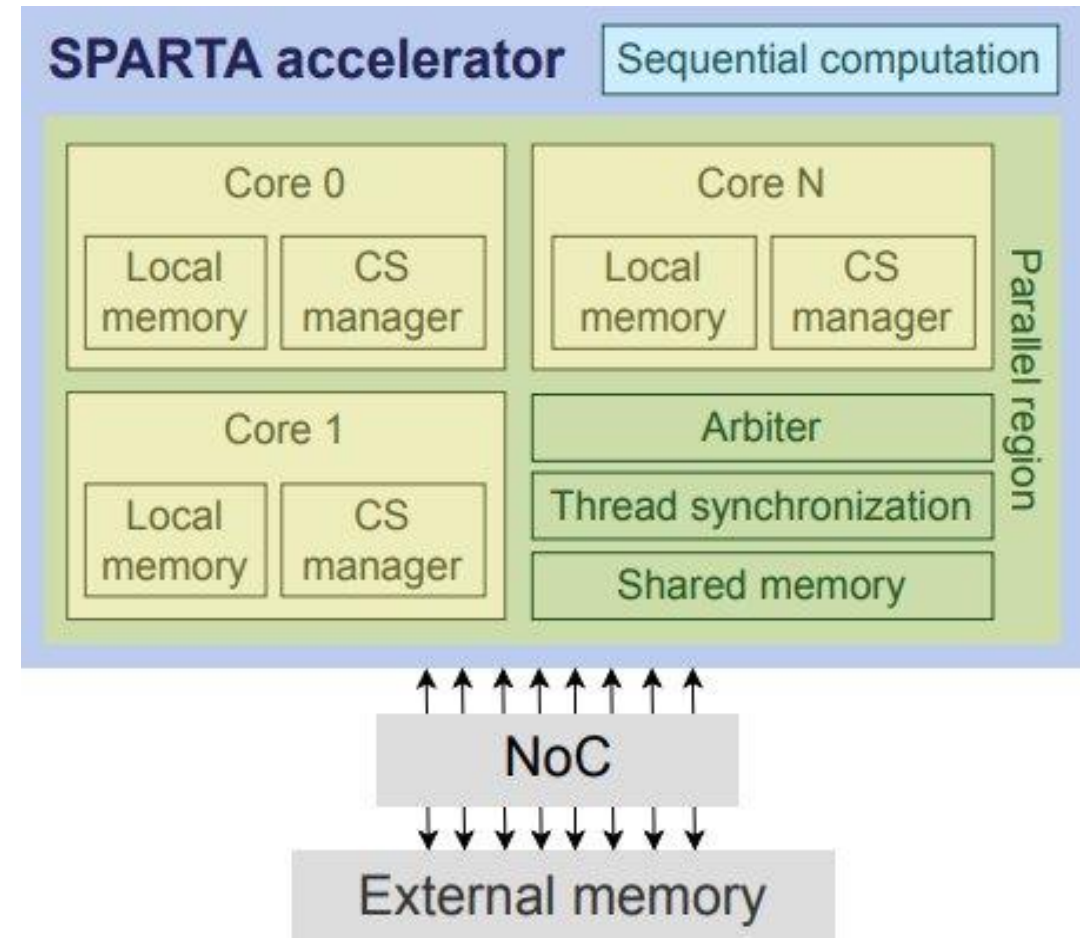
- Each core implements a software function
- It has its own local memory (easy to access)
- It has a local Context_Switch manager module
- Multiple threads can execute on the same core: when an external operation requires multiple cycles the context active in the core changes increasing the utilization of the hardware resources and hiding the latency of the memory.



SPARTA architecture

Each parallel region can contain multiple cores:

- SPARTA adds hardware to perform the arbitration of memory channels and synchronization between threads (critical sections, barriers...)
- It allocates inside the parallel region the variable shared between the different cores.
- SPARTA supports nested parallel region

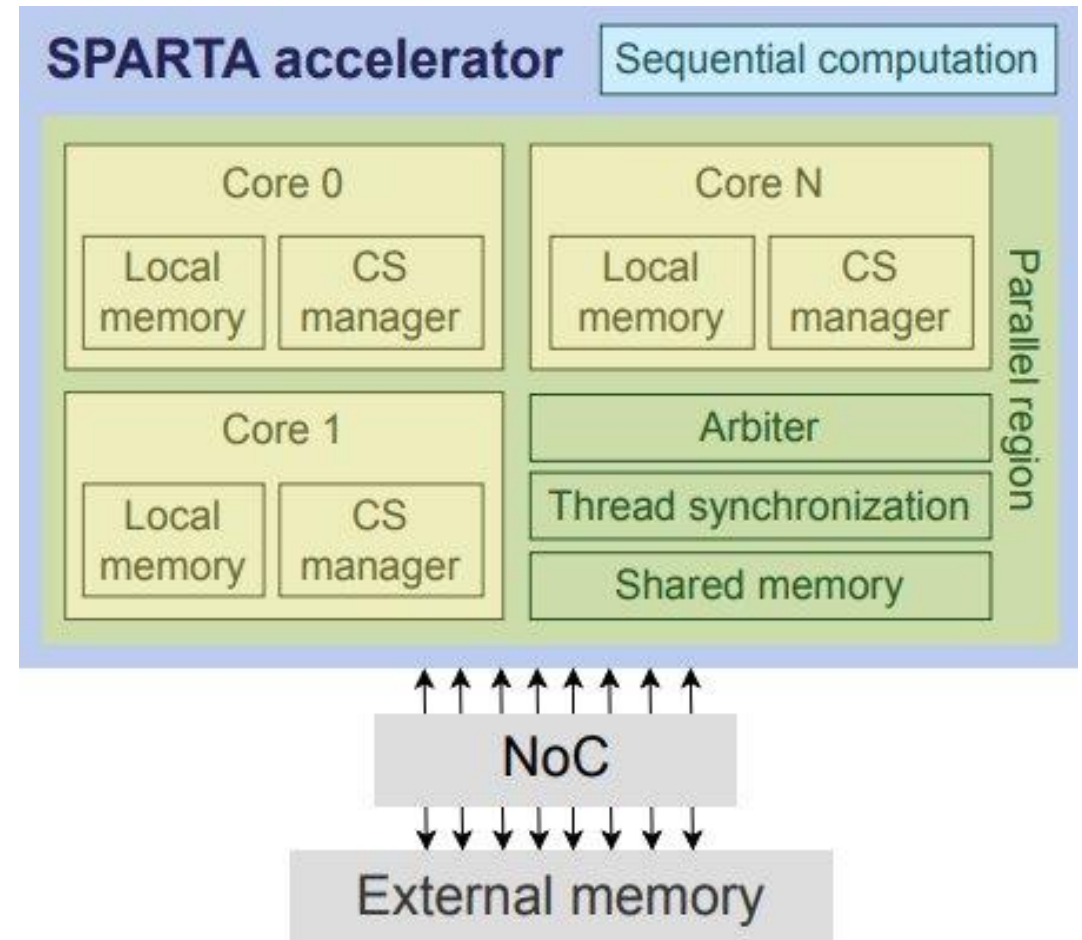


SPARTA architecture

Each parallel region is contained inside a sequential region and can have multiple channels towards the memory outside of the FPGA.

SPARTA can use AXI4 protocol and an HBM interconnect to exchange data with a banked memory.

An ad-hoc NoC is used to handle the requests and send them to the right bank.



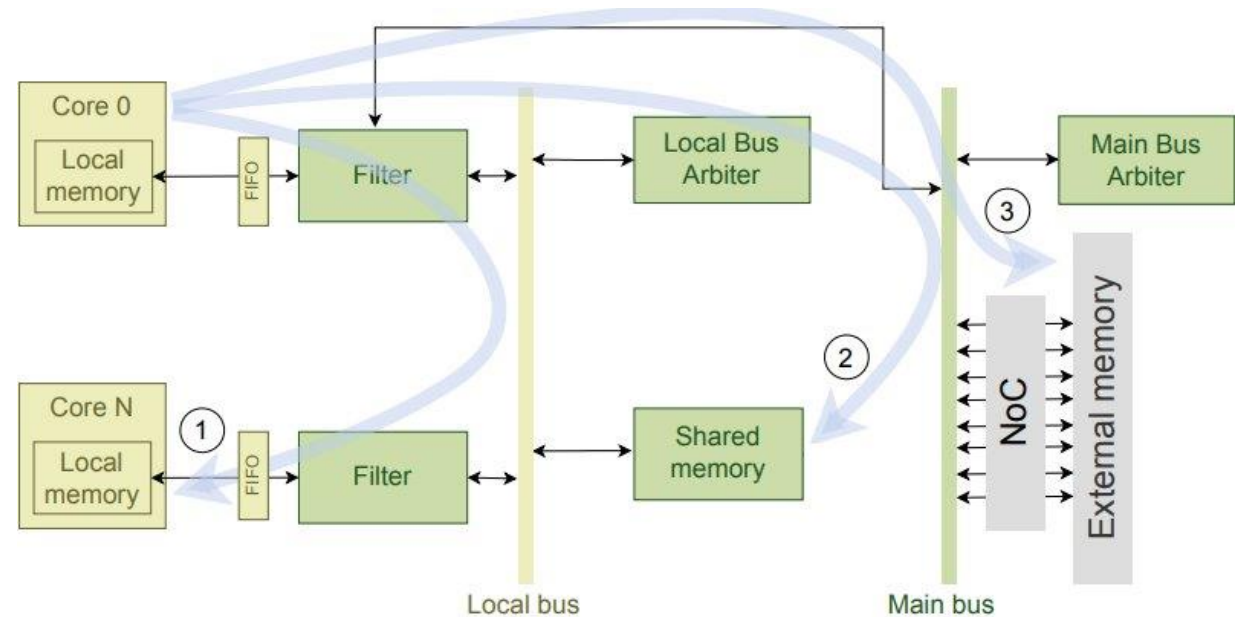
SPARTA architecture - memory

SPARTA can allocate variables in three locations:

- Inside the cores
- In the parallel region
- In the sequential region

SPARTA uses a memory bus with filters to detect which memory contains the requested variable.

The local bus can operate independently from the main memory bus.



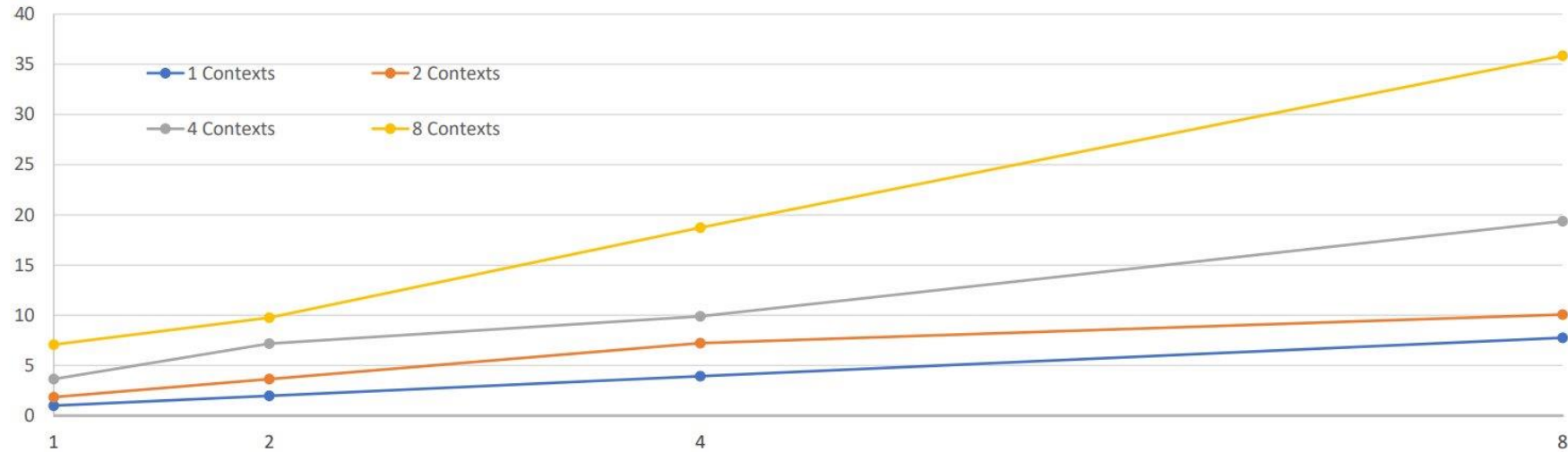
Results

SPARTA architecture achieves better results with irregular applications when sharing multiple contexts on the same core hides the memory access latency. We have tested it using different algorithm taken from the Graph Algorithm Platform Benchmark Suite (GAPBS)

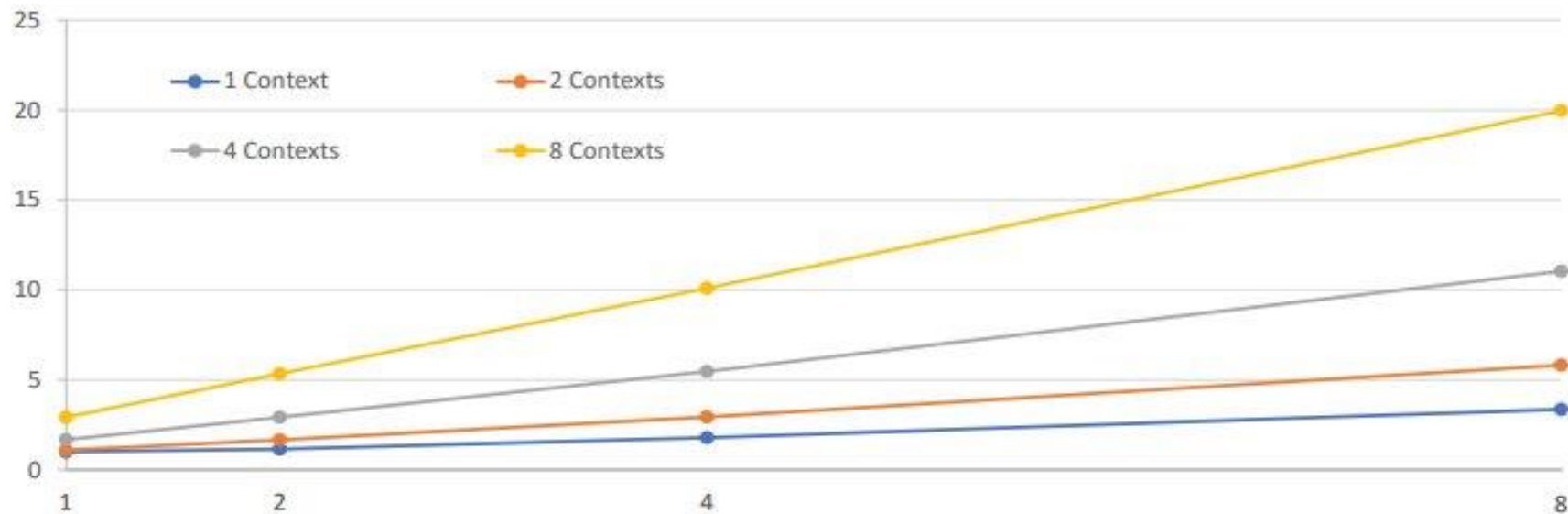


<https://github.com/sbeamer/gapbs>

Results



Speed up of SPARTA over the sequential baseline for the Connected Components benchmark.



Speed up of SPARTA over the sequential baseline for the Triangle Count benchmark.

Questions



<https://panda.dei.polimi.it>

<https://github.com/ferrandi/PandA-bambu>

This work has been partially supported by the Spoke 1 "FutureHPC \ & BigData" of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 - Next Generation EU (NGEU).