AFF3CT Runtime × julia – New Features & Roadmap

2nd AFF3CT User Day

Adrien Cassagne[†], Maxime Millet[†], Julien Sopena[†] and Alix Munier[†]

[†] *Sorbonne University, LIP6, CNRS*, Paris, France

November 28, 2023

Table of Contents

- Introduction
- New Features
- Roadmap
- Conclusion

Remainder of Previous AFF3CT User Days

- **AFF3CT** split into 3 open source projects (MIT license)
  - **AFF3CT**[1]: Library & simulator for **error correcting codes**
    - GitHub: 426 ★ – 135 forks – 90k lines of code
  - **AFF3CT-core**[2]: Dataflow **DSEL** & **multi-threaded** runtime
    - GitHub: 1 ★ – 2 forks – 20k lines of code
  - **MIPP**[3]: **Portability** & expressiveness for CPU **SIMD instructions**
    - GitHub: 440 ★ – 85 forks – 30k lines of code

[1] A. Cassagne, O. Hartmann, M. Léonardon, K. He, C. Leroux, R. Tajan, O. Aumage, D. Barthou, T. Tonnellier, V. Pignoly, B. Le Gal, and C. Jégo. "AFF3CT: A Fast Forward Error Correction Toolbox!" In: *Elsevier SoftwareX* 10 (Oct. 2019), p. 100345. DOI: `10.1016/j.softx.2019.100345`.

[2] A. Cassagne, R. Tajan, O. Aumage, D. Barthou, C. Leroux, and C. Jégo. "A DSEL for High Throughput and Low Latency Software-Defined Radio on Multicore CPUs". In: *Wiley Concurrency and Computation: Practice and Experience (CCPE)* 35.23 (July 2023), e7820. DOI: `10.1002/cpe.7820`.

[3] A. Cassagne, O. Aumage, D. Barthou, C. Leroux, and C. Jégo. "MIPP: A Portable C++ SIMD Wrapper and its use for Error Correction Coding in 5G Standard". In: *Workshop on Programming Models for SIMD/Vector Processing (WPMVP)*. Vösendorf/Wien, Austria: ACM, Feb. 2018. DOI: `10.1145/3178433.3178435`.

Main Purposes of this Talk

1. Give you an overview of the **recently introduced features**
2. Talk about what is next: special focus on **AFF3CT-core** and julia
3. Discuss with you about the **directions to take**

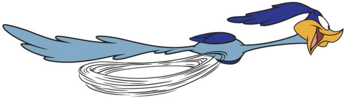
Table of Contents

2 New Features

- ▶ Introduction
- ▶ New Features
- ▶ Roadmap
- ▶ Conclusion

Single Instruction Multiple Data with MIPP

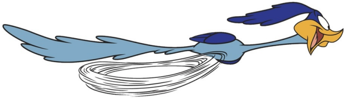
- **MIPP** enables
  - **Efficient implementations**
  - **Portability** over the most common architectures
  - Code **readability** compared to intrinsic functions



MIPP MIPP!

Single Instruction Multiple Data with MIPP

- **MIPP** enables
  - **Efficient implementations**
  - **Portability** over the most common architectures
  - Code **readability** compared to intrinsic functions

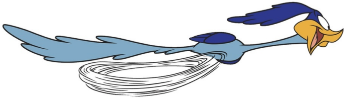


MIPP MIPP!

- **New features**
  - **Unsigned integers** support (relevant for some signal processing algorithms)
  - Partial **support of SVE**
    - SVE Length Specific
    - Most common operations for floating-point numbers
    - SIMD ISA in ARMv9 and in Fujitsu A64FX CPUs (in Fugaku, the World n°2 Supercomputer)
  - Working on a **code generator** (L. Dendani's 6 months internship @ IFPEN)

Single Instruction Multiple Data with MIPP

- **MIPP** enables
  - **Efficient implementations**
  - **Portability** over the most common architectures
  - Code **readability** compared to intrinsic functions



MIPP MIPP!

- **New features**
  - **Unsigned integers** support (relevant for some signal processing algorithms)
  - Partial **support of SVE**
    - SVE Length Specific
    - Most common operations for floating-point numbers
    - SIMD ISA in ARMv9 and in Fujitsu A64FX CPUs (in Fugaku, the World n°2 Supercomputer)
  - Working on a **code generator** (L. Dendani's 6 months internship @ IFPEN)
- **Collaborations**
  - IFP Energies Nouvelles (IFPEN)
  - Inria Bordeaux
    - **Open position for a 6 months internship** → Code generation, SVE & RVV

AFF3CT-core for Streaming Applications

“At the creation of the Universe, AFF3CT & AFF3CT-core was a single project.”

- **AFF3CT-core has been extracted from AFF3CT** as is it no longer specific to digital communications
  - Still, AFF3CT-core is a strong dependency in AFF3CT
  - Enable to **work in an asynchronous way on both projects**

AFF3CT-core for Streaming Applications

“At the creation of the Universe, AFF3CT & AFF3CT-core was a single project.”

- **AFF3CT-core has been extracted from AFF3CT** as is it no longer specific to digital communications
  - Still, AFF3CT-core is a strong dependency in AFF3CT
  - Enable to **work in an asynchronous way on both projects**
- A DSEL and a runtime to **support a large range of applications**
  - Digital communications: DVB-S2 transceiver, DVB-RCS2 transceiver, ...
  - Computer vision: real-time meteor detection systems
  - Post-quantum cryptography: on going work of Andrea Lesavourey
  - DNN inference: Enrique Galvez’s 6 months internship starting February’24 (co-supervised with Alix Munier @ LIP6)
  - ➔ **Streaming applications!**

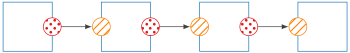
AFF3CT-core – Definitions

2 New Features

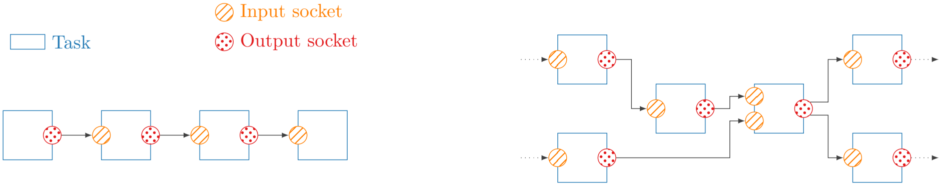
▭ Task

▨ Input socket

⁘ Output socket



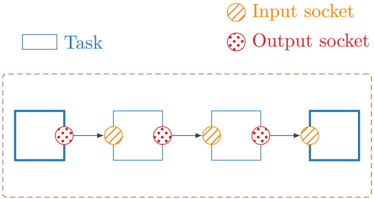
AFF3CT-core – Definitions



- **Directed graphs** are supported **to map a wide range of apps**

AFF3CT-core – Definitions

2 New Features

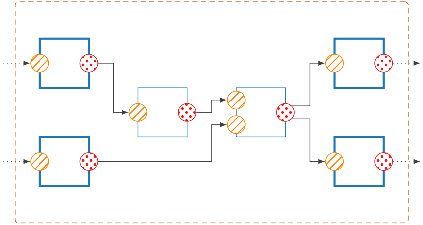


▭ Task

▨ Input socket

⁘ Output socket

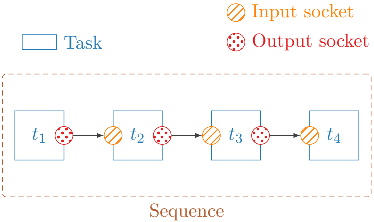
Sequence

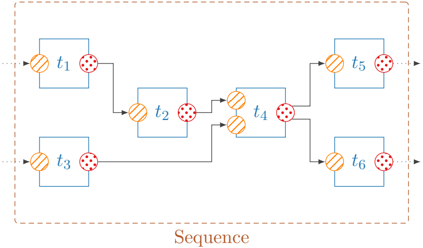


Sequence

- **Directed graphs** are supported **to map a wide range of apps**
- A *sequence* is built from an initial and a final list of tasks

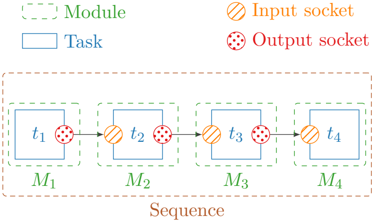
AFF3CT-core – Definitions

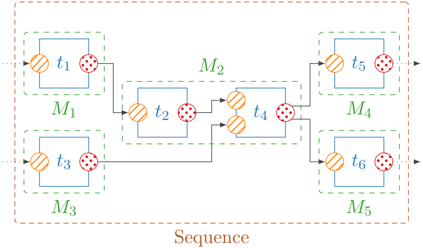




- **Directed graphs** are supported **to map a wide range of apps**
- A *sequence* is built from an initial and a final list of tasks
- Tasks execution order (scheduling) is determined by the user binding

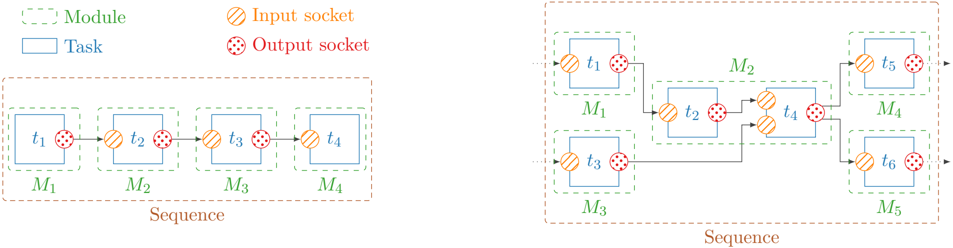
AFF3CT-core – Definitions





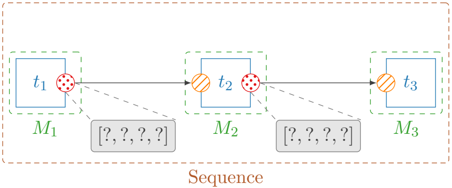
- **Directed graphs** are supported **to map a wide range of apps**
- A *sequence* is built from an initial and a final list of tasks
- Tasks execution order (scheduling) is determined by the user binding
- **States** are contained in *modules* (= C++ classes)

AFF3CT-core – Definitions



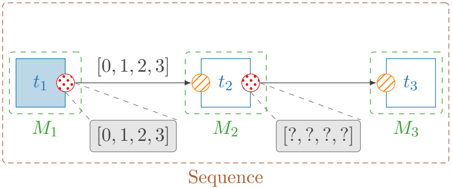
- **Directed graphs** are supported **to map a wide range of apps**
- A *sequence* is built from an initial and a final list of tasks
- Tasks execution order (scheduling) is determined by the user binding
- **States** are contained in *modules* (= C++ classes)
- One task execution is enough to run dependent tasks (**single rate SDF**)

AFF3CT-core – Forward Socket – Presentation



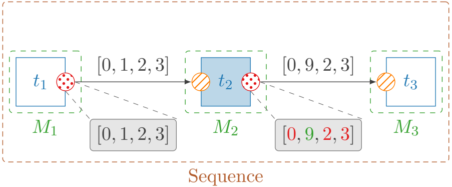
- Data are automatically allocated in the output sockets (see gray rectangles)

AFF3CT-core – Forward Socket – Presentation



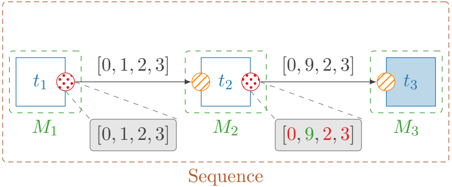
- Data are automatically allocated in the output sockets (see gray rectangles)
- Let's assume that t_2 only modify the second value of its input socket

AFF3CT-core – Forward Socket – Presentation



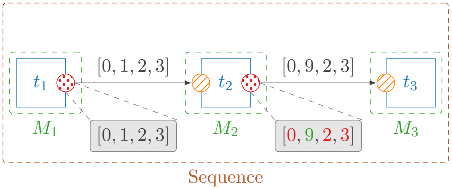
- Data are automatically allocated in the output sockets (see gray rectangles)
- Let's assume that t_2 only modify the second value of its input socket
  - "0", "2" and "3" are copied into t_2 output socket and "9" value replaces "1"

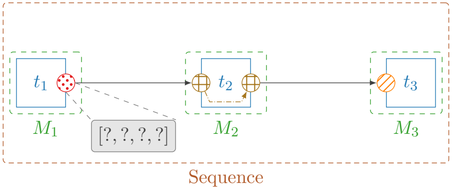
AFF3CT-core – Forward Socket – Presentation



- Data are automatically allocated in the output sockets (see gray rectangles)
- Let's assume that t_2 only modify the second value of its input socket
  - "0", "2" and "3" are copied into t_2 output socket and "9" value replaces "1"
  - This is highly inefficient!

AFF3CT-core – Forward Socket – Presentation

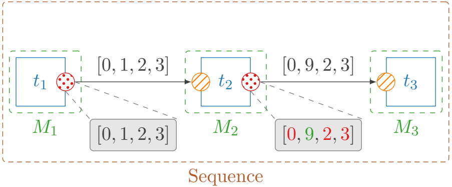


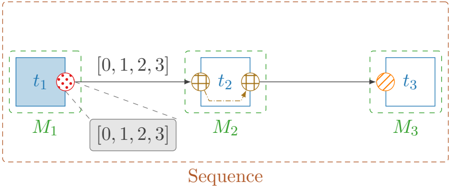


- Data are automatically allocated in the output sockets (see gray rectangles)
- Let's assume that t_2 only modify the second value of its input socket
  - "0", "2" and "3" are copied into t_2 output socket and "9" value replaces "1"
  - This is highly inefficient!

- Forward socket: at the same time an input and output socket (read+write)
  - There is NO data allocation

AFF3CT-core – Forward Socket – Presentation

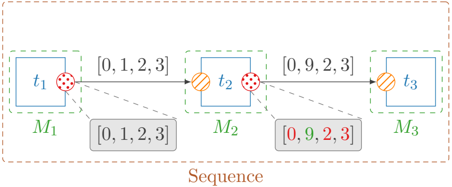




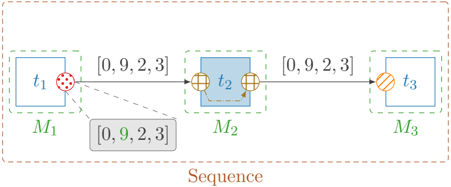
- Data are automatically allocated in the output sockets (see gray rectangles)
- Let's assume that t_2 only modify the second value of its input socket
  - "0", "2" and "3" are copied into t_2 output socket and "9" value replaces "1"
  - This is highly inefficient!

- Forward socket: at the same time an input and output socket (read+write)
  - There is NO data allocation
- We propose a new implementation of t_2 with a forward socket

AFF3CT-core – Forward Socket – Presentation

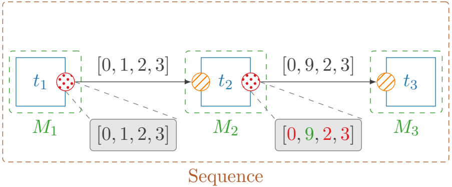


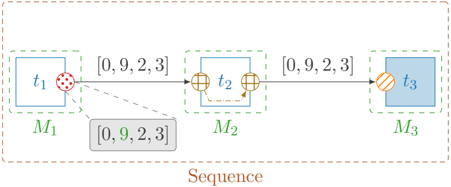
- Data are automatically allocated in the output sockets (see gray rectangles)
- Let's assume that t_2 only modify the second value of its input socket
  - "0", "2" and "3" are copied into t_2 output socket and "9" value replaces "1"
  - This is highly inefficient!



- Forward socket: at the same time an input and output socket (read+write)
  - There is NO data allocation
- We propose a new implementation of t_2 with a forward socket
  - t_1 output socket is modified in-place ("1" becomes "9")

AFF3CT-core – Forward Socket – Presentation





- Data are automatically allocated in the output sockets (see gray rectangles)
- Let's assume that t_2 only modify the second value of its input socket
  - "0", "2" and "3" are copied into t_2 output socket and "9" value replaces "1"
  - This is highly inefficient!

- Forward socket: at the same time an input and output socket (read+write)
  - There is NO data allocation
- We propose a new implementation of t_2 with a forward socket
  - t_1 output socket is modified in-place ("1" becomes "9")
  - This is efficient and cache-friendly!

AFF3CT-core – Forward Socket – Summary

- **New concept** recently added to AFF3CT-core
  - **Master 1 project** with two students (Yacine Idouar & Nourdinne Hammachi)
  - Yacine Idouar's **Master 1 internship** (June to July'23)
  - Co-supervised with Julien Sopena @ LIP6
- **Operate in parallel contexts** across pipeline stages & sequence replications
- **Continuous integration** over extensive unitary testing & **documentation**
- **Proven to work** on a "real project": ×2 speedup for meteor detection

AFF3CT-core – Control Flow & Pipeline

- **Dynamic control flow**
  - Not common in dataflow DSL
  - Also known as *feedback loop*
    - Still, in the DSEL it is more generic as **dynamic conditions and switch-cases** are also supported
  - Useful in many cases
    - Digital communications: turbo demodulation
    - Computer vision: iterative optical flow
    - DNN: feedback graphs
    - And so on
  - Avoid unnecessary unrolling
    - Can be seen as a compression
  - Enable **dynamic early exit**
    - New optimization opportunities
    - **Static graph but dynamic path**

AFF3CT-core – Control Flow & Pipeline

- **Dynamic control flow**
  - Not common in dataflow DSL
  - Also known as *feedback loop*
    - Still, in the DSEL it is more generic as **dynamic conditions and switch-cases** are also supported
  - Useful in many cases
    - Digital communications: turbo demodulation
    - Computer vision: iterative optical flow
    - DNN: feedback graphs
    - And so on
  - Avoid unnecessary unrolling
    - Can be seen as a compression
  - Enable **dynamic early exit**
    - New optimization opportunities
    - **Static graph but dynamic path**

- **New features**
  - **Control flow inside a pipeline stage**
    - Tested in continuous integration of comprehensive cases
    - Documented
  - **Improved error management**
    - Control flow must be within a stage
    - Error message if the data binding is wrong
  - Master 1 internship (2 months)
    - Nourdinne Hammachi
    - Co-supervised with Julien Sopena @ LIP6
- Still some **limitations**
  - A pipeline stage cannot start or end with a Switcher task (`select` or `commute`)
  - `relay` task can overcome this issue
    - At the cost of a useless copy...

AFF3CT-core – Miscellaneous

- **2D socket**
  - — Memory is still allocated in a **contiguous way**
  - — But an additional row buffer is allocated for the 2nd dimension in the socket
  - — Addressed in the previous buffer need to be recomputed each time a task is triggered
  - — 3D socket is considered in the future

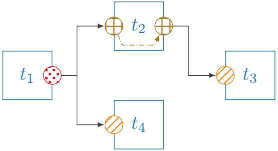
```
Stateless foo(); // create a module
Task &t = foo.create_tsk("bar"); // create a task
// create a 2D socket (8 rows and 8 cols = 64 elmts)
size_t si = foo.create_2d_sck_in<int>(t, "in", 8, 8);
// code to execute when the 'bar' task is called
foo.create_codelet(t, [si](Module &m, Task &tsk) {
  // get a input 2D data pointer
  const int** img =
    tsk[si].get_2d_dataptr<const int>();
  volatile int sum = 0;
  for (size_t y = 0; y < 8; y++)
    for (size_t x = 0; x < 8; x++)
      sum += img[y][x];
  return status_t::SUCCESS;
});
```

AFF3CT-core – Miscellaneous

- **2D socket**
  - Memory is still allocated in a **contiguous way**
  - But an additional row buffer is allocated for the 2nd dimension in the socket
  - Addressed in the previous buffer need to be recomputed each time a task is triggered
  - 3D socket is considered in the future

```
Stateless foo(); // create a module
Task &t = foo.create_tsk("bar"); // create a task
// create a 2D socket (8 rows and 8 cols = 64 elmts)
size_t si = foo.create_2d_sck_in<int>(t, "in", 8, 8);
// code to execute when the 'bar' task is called
foo.create_codelet(t, [si](Module &m, Task &tsk) {
  // get a input 2D data pointer
  const int** img =
    tsk[si].get_2d_dataptr<const int>();
  volatile int sum = 0;
  for (size_t y = 0; y < 8; y++)
    for (size_t x = 0; x < 8; x++)
      sum += img[y][x];
  return status_t::SUCCESS;
});
```

- **Task to task binding**
  - Allow to **specify dependencies** between tasks **more precisely**
  - Required in some cases
    - Non-explicit dependencies (= not described through sockets binding)
    - Forward socket can change the data of an output socket ⇒ the execution order can modify final result

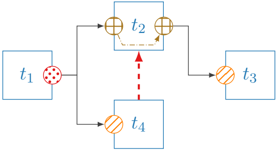


AFF3CT-core – Miscellaneous

- **2D socket**
  - Memory is still allocated in a **contiguous way**
  - But an additional row buffer is allocated for the 2$^{\text{nd}}$ dimension in the socket
  - Addressed in the previous buffer need to be recomputed each time a task is triggered
  - 3D socket is considered in the future

```cpp
Stateless foo(); // create a module
Task &t = foo.create_tsk("bar"); // create a task
// create a 2D socket (8 rows and 8 cols = 64 elmts)
size_t si = foo.create_2d_sck_in<int>(t, "in", 8, 8);
// code to execute when the 'bar' task is called
foo.create_codelet(t, [si](Module &m, Task &tsk) {
  // get a input 2D data pointer
  const int** img =
    tsk[si].get_2d_dataptr<const int>();
  volatile int sum = 0;
  for (size_t y = 0; y < 8; y++)
    for (size_t x = 0; x < 8; x++)
      sum += img[y][x];
  return status_t::SUCCESS;
});
```

- **Task to task binding**
  - Allow to **specify dependencies** between tasks **more precisely**
  - Required in some cases
    - Non-explicit dependencies (= not described through sockets binding)
    - Forward socket can change the data of an output socket ⇒ the execution order can modify final result



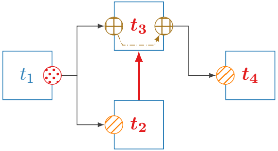
AFF3CT-core – Miscellaneous

2 New Features

- **2D socket**
  - — Memory is still allocated in a **contiguous way**
  - — But an additional row buffer is allocated for the 2nd dimension in the socket
  - — Addressed in the previous buffer need to be recomputed each time a task is triggered
  - — 3D socket is considered in the future

```
Stateless foo(); // create a module
Task &t = foo.create_tsk("bar"); // create a task
// create a 2D socket (8 rows and 8 cols = 64 elmts)
size_t si = foo.create_2d_sck_in<int>(t, "in", 8, 8);
// code to execute when the 'bar' task is called
foo.create_codelet(t, [si](Module &m, Task &tsk) {
  // get a input 2D data pointer
  const int** img =
    tsk[si].get_2d_dataptr<const int>();
  volatile int sum = 0;
  for (size_t y = 0; y < 8; y++)
    for (size_t x = 0; x < 8; x++)
      sum += img[y][x];
  return status_t::SUCCESS;
});
```

- **Task to task binding**
  - — Allow to **specify dependencies** between tasks **more precisely**
  - — Required in some cases
    - ○ Non-explicit dependencies (= not described through sockets binding)
    - ○ Forward socket can change the data of an output socket ⇒ the execution order can modify final result



Knowledge Transfer

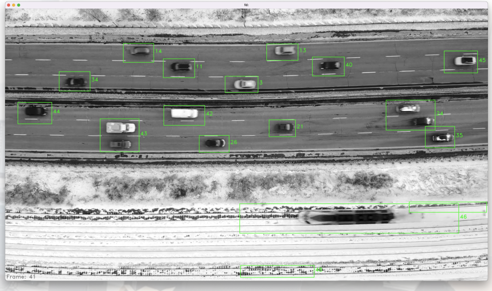
2 New Features

- **Teaching MIPP**
  - Polytech Sorbonne Engineering School – 3rd year
    - *Électronique et Informatique – Systèmes Embarqués*
  - Sorbonne University – Master 2 SESI
    - *Systèmes Électroniques et Systèmes Informatiques*

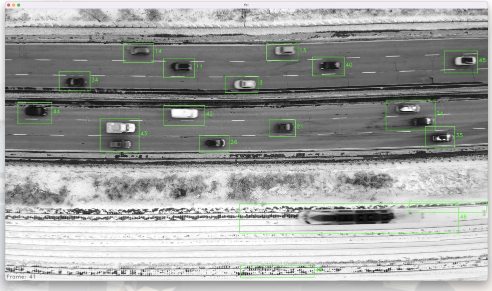
Knowledge Transfer

- **Teaching MIPP**
  - — Polytech Sorbonne Engineering School – 3rd year
    - ○ *Électronique et Informatique – Systèmes Embarqués*
  - — Sorbonne University – Master 2 SESI
    - ○ *Systèmes Électroniques et Systèmes Informatiques*

- **Teaching AFF3CT**
  - — Sorbonne University – SESI M2
    - ○ Motion detection and tracking on embedded systems



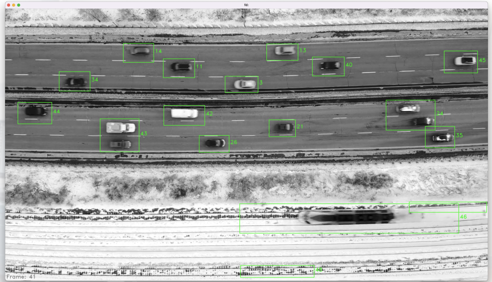
Knowledge Transfer

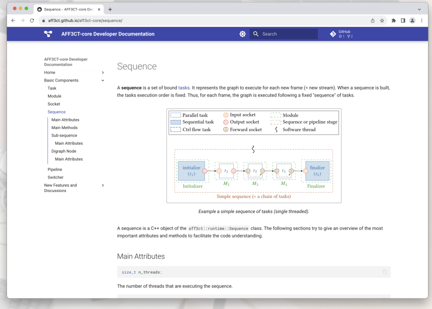


- **Teaching MIPP**
  - — Polytech Sorbonne Engineering School – 3rd year
    - ○ *Électronique et Informatique – Systèmes Embarqués*
  - — Sorbonne University – Master 2 SESI
    - ○ *Systèmes Électroniques et Systèmes Informatiques*
- **Teaching AFF3CT**
  - — Sorbonne University – SESI M2
    - ○ Motion detection and tracking on embedded systems
- **Materials available online** for the community
  - — `https://lip6.fr/adrien.cassagne/#teaching`

Knowledge Transfer

- **Teaching MIPP**
  - Polytech Sorbonne Engineering School – 3rd year
    - *Électronique et Informatique – Systèmes Embarqués*
  - Sorbonne University – Master 2 SESI
    - *Systèmes Électroniques et Systèmes Informatiques*
- **Teaching AFF3CT**
  - Sorbonne University – SESI M2
    - Motion detection and tracking on embedded systems
- **Materials available online** for the community
  - `https://lip6.fr/adrien.cassagne/#teaching`
- New **AFF3CT-core developer documentation**
  - `https://aff3ct.github.io/aff3ct-core/`




Sequence - AFF3CT-core Dev
aff3ct.github.io/aff3ct-core/sequence/
AFF3CT-core Developer Documentation
Search
GitHub
☆1 ⑂2
AFF3CT-core Developer Documentation
Home
Basic Components
Task
Module
Socket
Sequence
Main Attributes
Main Methods
Sub-sequence
Main Attributes
Digraph Node
Main Attributes
Pipeline
Switcher
New Features and Discussions

Sequence

A **sequence** is a set of bound tasks. It represents the graph to execute for each new frame (= new stream). When a sequence is built, the tasks execution order is fixed. Thus, for each frame, the graph is executed following a fixed "sequence" of tasks.

Example a simple sequence of tasks (single threaded).

A sequence is a C++ object of the `aff3ct::runtime::Sequence` class. The following sections try to give an overview of the most important attributes and methods to facilitate the code understanding.

Main Attributes

```
size_t n_threads;
```

The number of threads that are executing the sequence.

Table of Contents

3 Roadmap

- Introduction
- New Features
- Roadmap
- Conclusion

Job Offer & New Recruit

- **Job offer**: Inria-DGA convention
  - Engineer or post-doc (**18 months**)
  - **Mission 1**: **Improve the DSEL** for high level languages
  - **Mission 2**: Use julia to **wrap AFF3CT-core** and to **enrich the library**

Job Offer & New Recruit

- **Job offer**: Inria-DGA convention
  - Engineer or post-doc (**18 months**)
  - **Mission 1**: **Improve the DSEL** for high level languages
  - **Mission 2**: Use julia to **wrap AFF3CT-core** and to **enrich the library**

- **New Recruit**: Maxime Millet
  - PhD in computer science @ LIP6
  - **Low level optimizations for SoCs**
    - SIMD for embedded architectures
    - Heterogeneous CPU/GPU implementations
  - **Optical flow and meteor detection**

AFF3CT-core × julia

3 Roadmap

- **Benefit from AFF3CT-core** in julia
  - DSEL for streaming applications
  - Multi-threaded runtime (pipeline & replications)

AFF3CT-core × julia

- **Benefit from AFF3CT-core** in julia
  - DSEL for streaming applications
  - Multi-threaded runtime (pipeline & replications)

- **Benefit from** julia in AFF3CT-core
  - High level language with **high expressiveness**
  - Simple type templatization
  - **Just in Time compilation**
    - LLVM passes to simplify AFF3CT-core

AFF3CT-core × julia

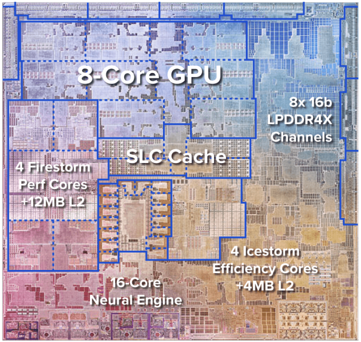
- **Benefit from AFF3CT-core** in julia
  - DSEL for streaming applications
  - Multi-threaded runtime (pipeline & replications)
- **Benefit from** julia in AFF3CT-core
  - High level language with **high expressiveness**
  - Simple type templatization
  - **Just in Time compilation**
    - LLVM passes to simplify AFF3CT-core
- **Study code vectorization in** julia
  - Compare explicit SIMD.jl versus MIPP
  - Are there some limitations?

AFF3CT-core × julia

- **Benefit from AFF3CT-core** in julia
  - DSEL for streaming applications
  - Multi-threaded runtime (pipeline & replications)
- **Benefit from** julia in AFF3CT-core
  - High level language with **high expressiveness**
  - Simple type templatization
  - **Just in Time compilation**
    - LLVM passes to simplify AFF3CT-core
- **Study code vectorization in** julia
  - Compare explicit SIMD.jl versus MIPP
  - Are there some limitations?
- **Enrich AFF3CT library** from code written in julia
  - Objective: **Simplify the AFF3CT contribution process**
  - Is it possible to use it in C++ and/or in Python?

AFF3CT-core – Heterogeneous Tasks Support

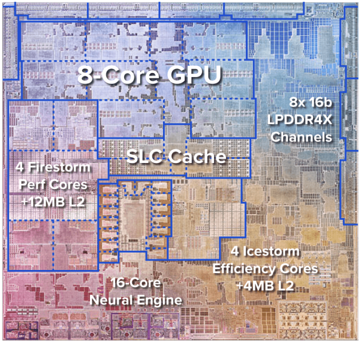
"Nowadays, processor manufacturers are releasing heterogeneous SoCs. On a same chip, we can find: energy efficient cores, performance cores, global memory and application specific accelerators like GPUs & NPUs."


8-Core GPU
8x 16b LPDDR4X Channels
SLC Cache
4 Firestorm Perf Cores +12MB L2
4 Icestorm Efficiency Cores +4MB L2
16-Core Neural Engine

AFF3CT-core – Heterogeneous Tasks Support

"Nowadays, processor manufacturers are releasing heterogeneous SoCs. On a same chip, we can find: energy efficient cores, performance cores, global memory and application specific accelerators like GPUs & NPUs."

- All these processing units **share the global memory**
  - — Take advantage of accelerators **without extra copies**
  - ➔ **New optimization opportunities** for streaming applications


8-Core GPU
8x 16b LPDDR4X Channels
SLC Cache
4 Firestorm Perf Cores +12MB L2
4 Icestorm Efficiency Cores +4MB L2
16-Core Neural Engine

AFF3CT-core – Heterogeneous Tasks Support

"Nowadays, processor manufacturers are releasing heterogeneous SoCs. On a same chip, we can find: energy efficient cores, performance cores, global memory and application specific accelerators like GPUs & NPUs."

- All these processing units **share the global memory**
  - — Take advantage of accelerators **without extra copies**
  - ➔ **New optimization opportunities** for streaming applications
- Add **heterogeneous tasks support** in AFF3CT-core
  - — Challenges: Enrich DSEL, memory allocations, scheduling
  - — Possible in julia : CUDA.jl, oneAPI, Apple GPUs
  - ➔ **Master 2 internship starting February'24** (co-supervised with Julien Sopena)

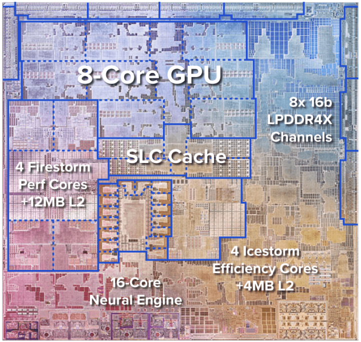

8-Core GPU
8x 16b LPDDR4X Channels
SLC Cache
4 Firestorm Perf Cores +12MB L2
4 Icestorm Efficiency Cores +4MB L2
16-Core Neural Engine

Table of Contents

4 Conclusion

- ▶ Introduction
- ▶ New Features
- ▶ Roadmap
- ▶ Conclusion

Final Words

4 Conclusion

- New features
  - MIPP: Unsigned integers, SVE support, working on a generator
  - AFF3CT-core: Forward sockets, control flow in pipeline stages
- Roadmap
  - Wrap & enrich AFF3CT with julia language
  - Heterogeneous tasks support in the runtime

Q&A

Thank you for listening!
Do you have any questions?

Bibliography

[1] A. Cassagne, O. Hartmann, M. Léonardon, K. He, C. Leroux, R. Tajan, O. Aumage, D. Barthou, T. Tonnellier, V. Pignoly, B. Le Gal, and C. Jégo. "AFF3CT: A Fast Forward Error Correction Toolbox!" In: *Elsevier SoftwareX* 10 (Oct. 2019), p. 100345. DOI: `10.1016/j.softx.2019.100345`.

[2] A. Cassagne, R. Tajan, O. Aumage, D. Barthou, C. Leroux, and C. Jégo. "A DSEL for High Throughput and Low Latency Software-Defined Radio on Multicore CPUs". In: *Wiley Concurrency and Computation: Practice and Experience (CCPE)* 35.23 (July 2023), e7820. DOI: `10.1002/cpe.7820`.

[3] A. Cassagne, O. Aumage, D. Barthou, C. Leroux, and C. Jégo. "MIPP: A Portable C++ SIMD Wrapper and its use for Error Correction Coding in 5G Standard". In: *Workshop on Programming Models for SIMD/Vector Processing (WPMVP)*. Vösendorf/Wien, Austria: ACM, Feb. 2018. DOI: `10.1145/3178433.3178435`.