WHICH METRICS FOR DEVELOPING LOW-COMPLEXITY VIDEO COMPRESSION SOLUTIONS?

Marco Mattavelli[1], Robert Turney[2]

[1]Ecole Polytechnique Fédérale de Lausanne (EPFL), Signal Processing Institute, Signal Processing Laboratory 3, CH-1015 Lausanne, Switzerland

[2]Xilinx Research Labs, San Jose,, CA, 95124 USA.

20 / 04 / 2005

Motivations

- The challenge has always been on higher and higher video compression performances with video quality appropriate for the application
- MPEG-1, MPEG-2, MPEG-4, AVC are the proof of the success of the approach
- Complexity:
  - to check that overall codecs complexity remained “realistic” for present and foreseeable future implementation technology,
  - to select between competing video tools providing equivalent coding efficiency the one presenting the lower implementation complexity.

Motivation

- Video at different (increasing) levels of resolutions is appearing in several existing or new application.
- There are applications for which a given available bandwidth exists, such as LANs or “system bus” interconnections, existing channels, low-cost cables etc etc….
- The objective is not to compress as much as possible, but as much as necessary to fit the relatively limited bandwidth.
- The criterion is to use the lowest possible resources.
- In other words when fixed a minimum target video quality and the compression ratio, the objective is to minimize coding/decoding costs.
- No current (MPEG and not) standard address expressly such optimization goal. The minimization criteria for the selection of compression tools to yield complete codecs should be “cost” efficiency or in other words low implementation complexity.

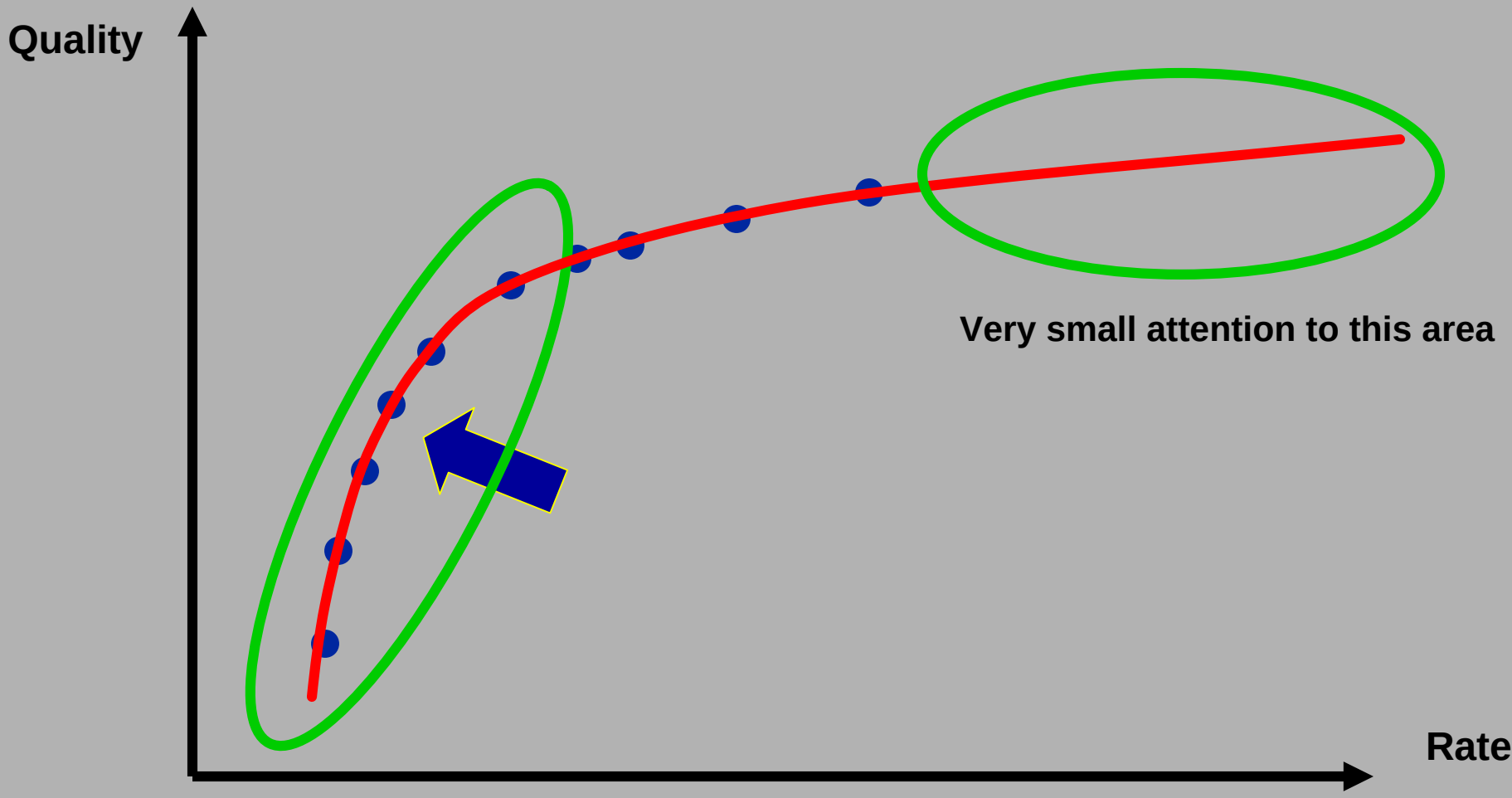
Implications

- Codecs should be as much as possible symmetrical regarding complexity (digital TV legacy).
- Encoding should be standardized (bounded in complexity).
- Selection of all tools and their combinations should be complexity driven.
- Transcoding and multiple generation effects should be carefully considered.

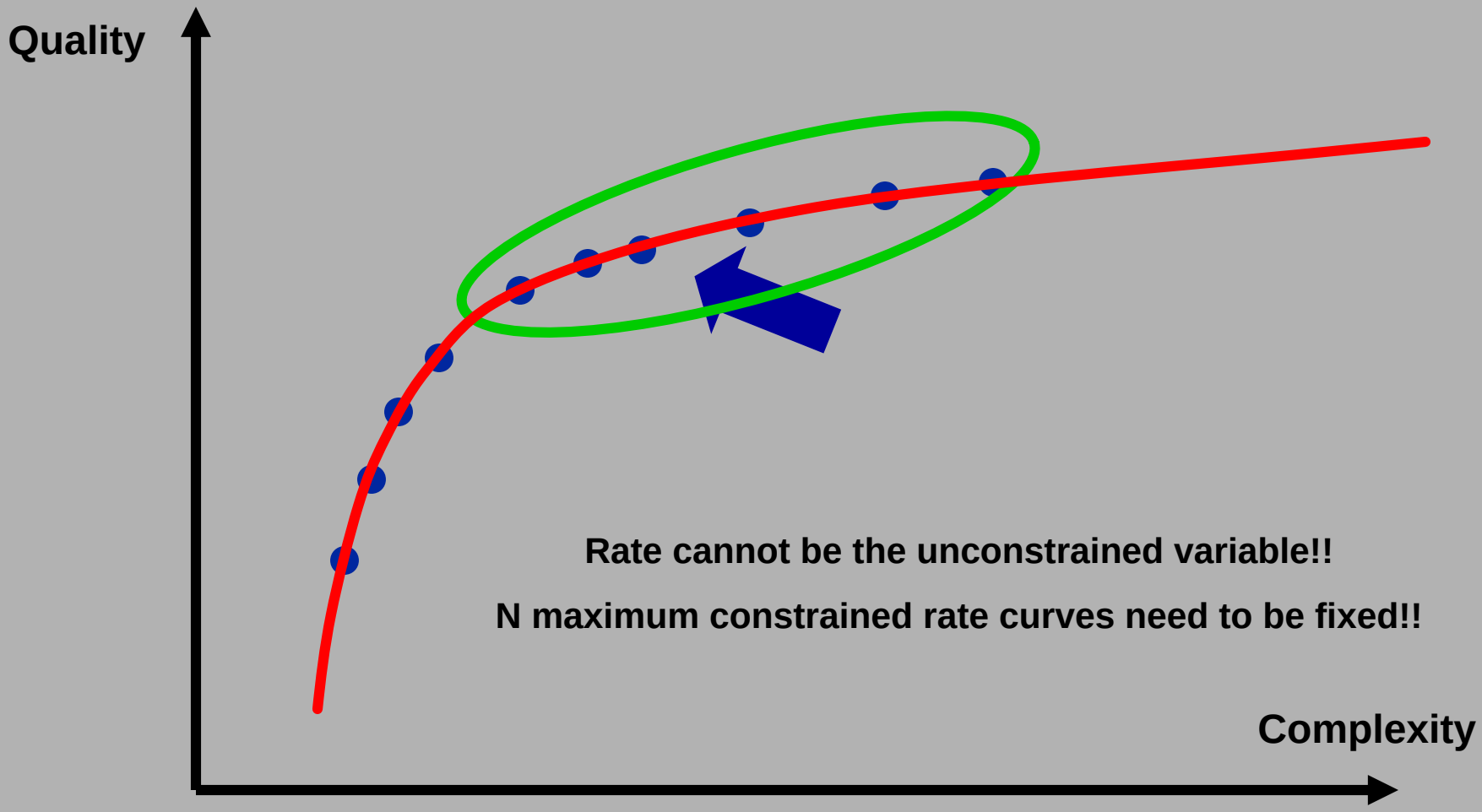
Why a standards is needed or would be useful?

- Interoperability between different manufacturers components.
- Duplication of efforts, everybody would need to profile existing standards for achieving low complexity
- Different “proprietary” home made codecs (patent owners will be protected?)

Current working points



Complexity oriented video coding



What is implementation complexity?

- How to measure implementation complexity?
  - Need a metric!
  - Need measurements tools!
  - Applied to which algorithm description/implementation?
- Real implementations?
  - Silicon area? Not practical!
  - Power dissipation? For which technology? Not practical!
  - Execution time on some processor? Which one, which version? Non representative for data flow complexity analysis!

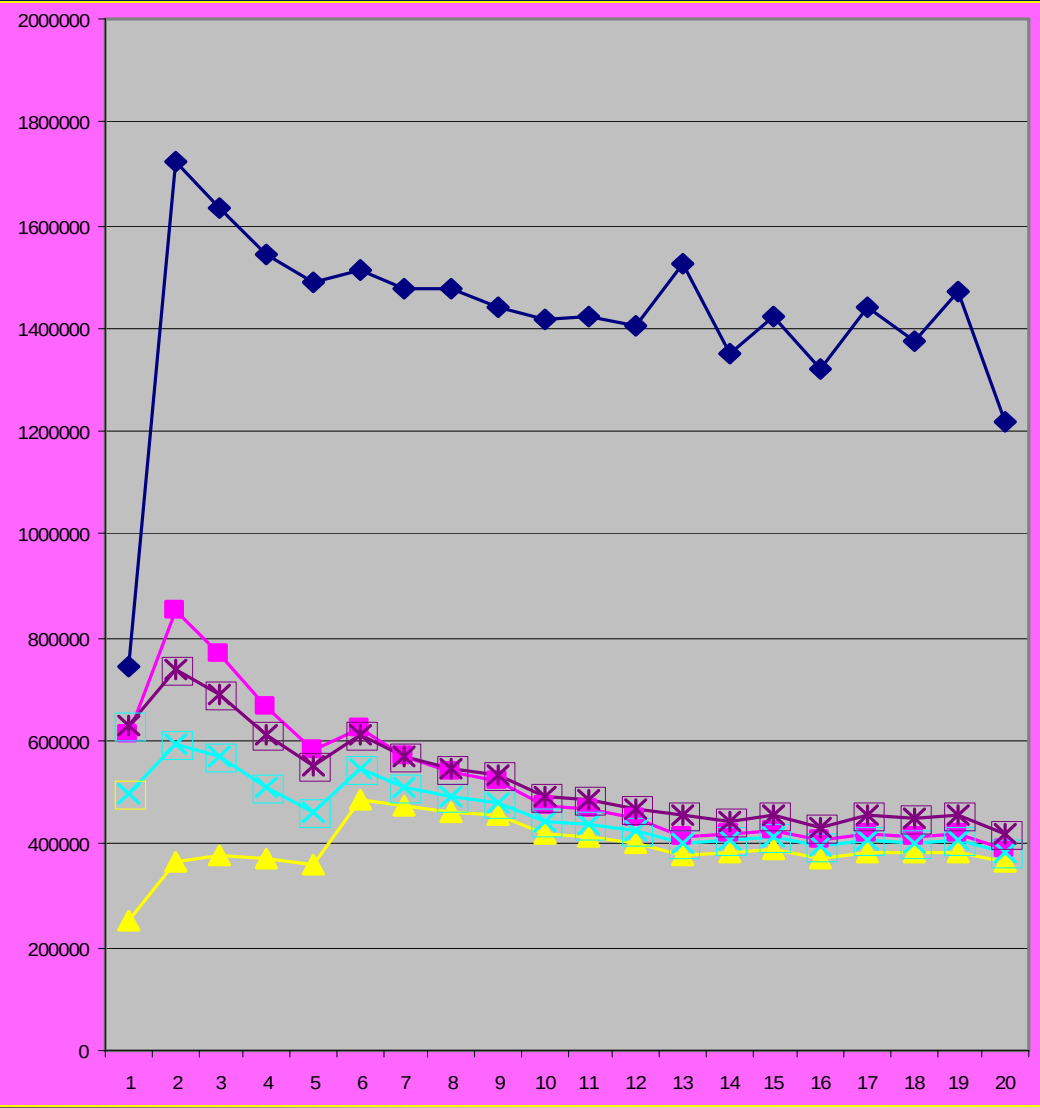
Complexity analysis

- Static code analysis:
  - Restricted code writing styles (no pointers or indirect addressing).
  - Worst case analysis not input data dependent.
- Dynamic code analysis beyond simple profilers:
  - Instruction profiling (IPROF)
  - C operator profiling, executed operators and data types (SIT)
  - Data transfer and storage estimation: measures of data transfers between variables, and memory size (ATOMIUM)

Exploration of design space

- Methodologies to explore the design space:
  - Moving upwards in the design flow
  - HDL or System C
  - Measures of architectural C descriptions (functions are architectural elements)
  - Measures of generic C descriptions

- Recent results shows that several metric measures can be extracted from the generic C algorithms descriptions



Blue: Reads AVC

Pink: Reads AVC rewritten buffer model 128bytes

Violet: Read AVC SIT model 128 bytes

Blue: Read AVC SIT model 256 bytes

Yellow: Read AVC SIT model 512 bytes

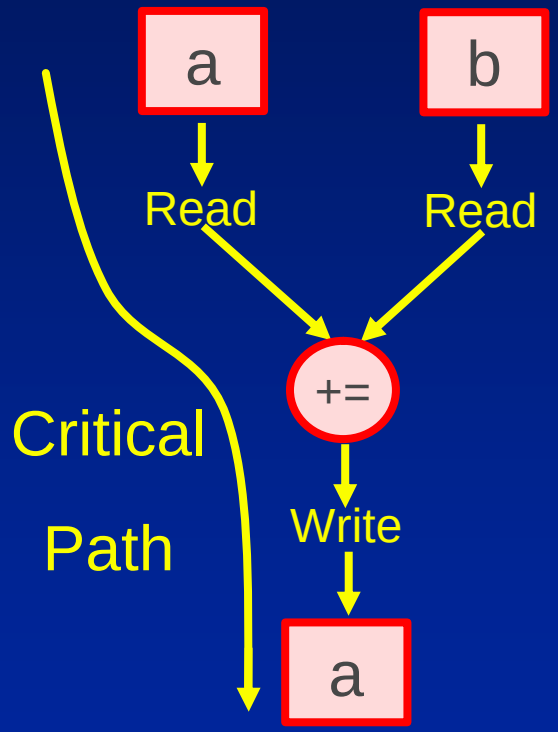
Critical Path Analysis

- Critical path represents the longest path in a circuit
- Useful metric for:
  - Determine the longest time sequence of weighted operation of data flow graph (and thus execution time, clock speed etc)
  - Parallel execution to evaluate parallelization potential
- Dynamic evaluation of critical path: true data dependencies, no explicit generation of the Data Flow Graph
  - Computed at runtime, taking into account the Data Flow Execution Graph and not the Static Data Flow Graph
  - Taking into account the weight of “really performed” operations
  - Using real input data

Critical Path Analysis

- Critical path on data dependencies

  Example: if a += b;



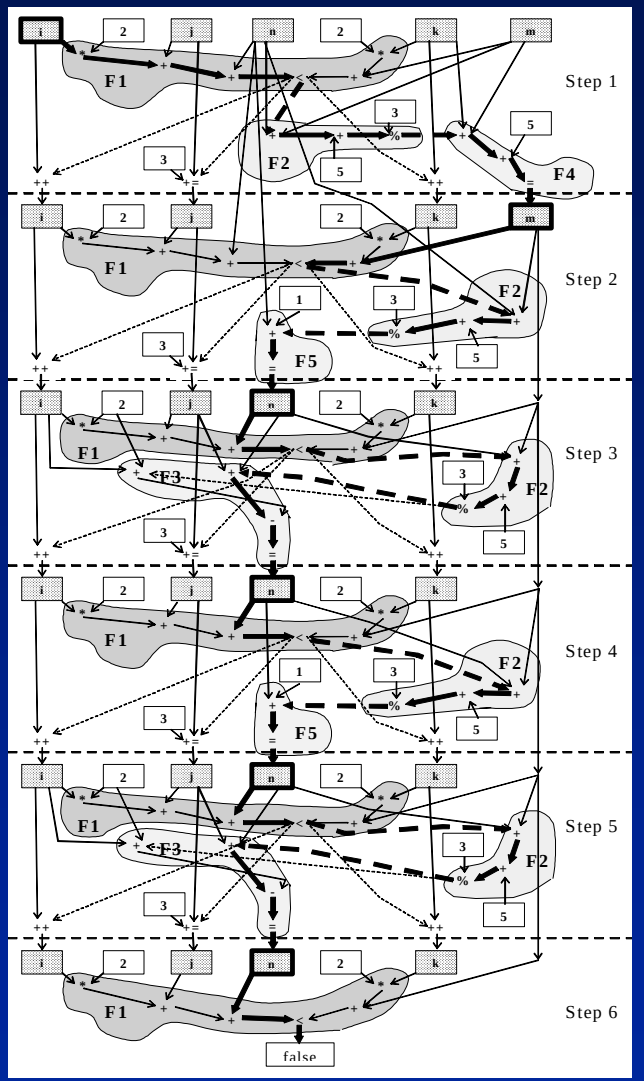
$$Cp = W(op) + Max(Cp(a),Cp(b))$$

Advantages:

- Dynamic evaluation: no coding style restrictions
- applicable on large algorithm, automatically instrumented by SIT
- Cp evaluation at variable/operation level

Critical Path Analysis

- This approach allows:
  - Evaluating overall critical path: lower parallelization bound (verifying optimization, estimating execution time…)
  - Working on parallelization potential, identifying highly parallelizable functions (ratio between overall complexity and critical path length)
  - Detecting functions that can be executed in parallel



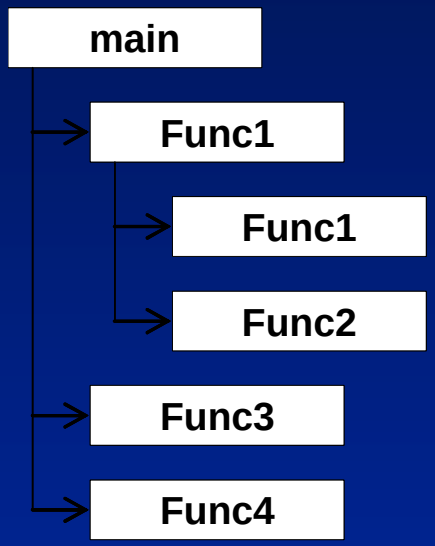
Critical Path Analysis

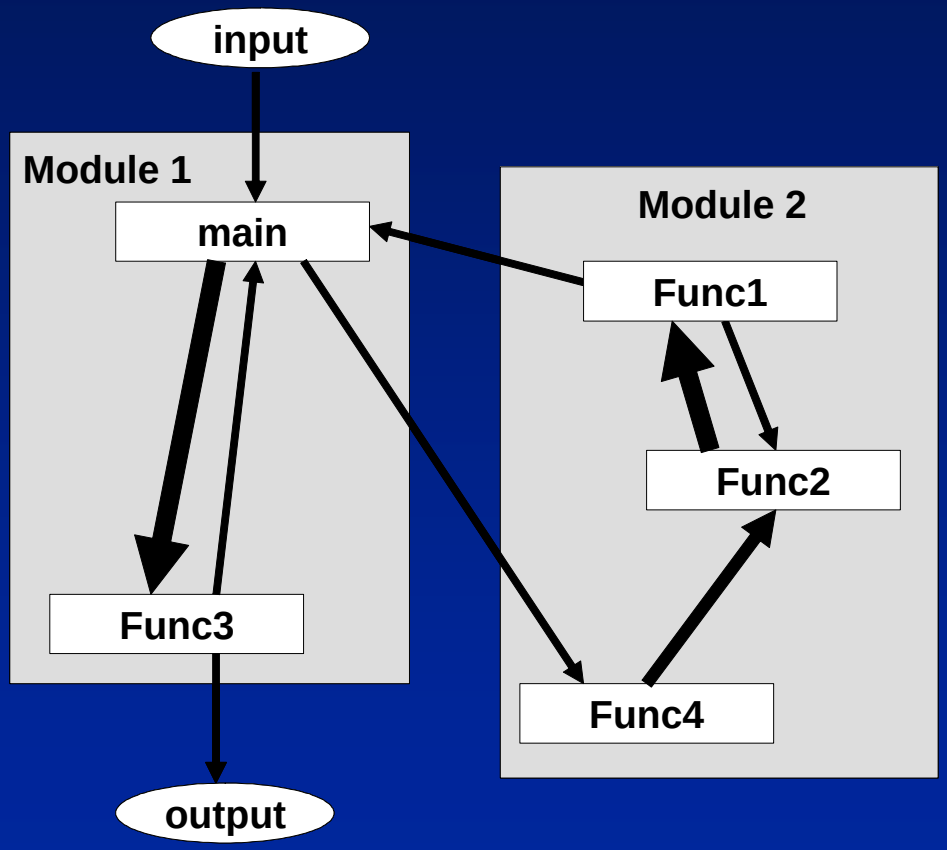

```
bool F1 (int n,int i,int j,int k,int m) {
        int r1=((n+j)+(i*2))<(k*2+m); return r1;
}
int F2 (int n, int m) {
        int r2=(n+m+1)%3;    return r2;
}
int F3 (int n, int i, int j) {  return n+j-i+2; }
int F4 (int m, int k) {  return m+k+5; }
int F5 (int n) {  return n+1; }

void main() {
        int i=2,j=5,k=11,n=0,m=0;
        for( ;  F1(n,i,j,k,m);  i++, j+=3, k++)
                switch (F2(n,m)) {
                        case 0: n=F3(n,i,j);    break;
                        case 1: m=F4(m,k);    break;
                        case 2: n=F5(n);        break;
                }
}
```

Data transfer

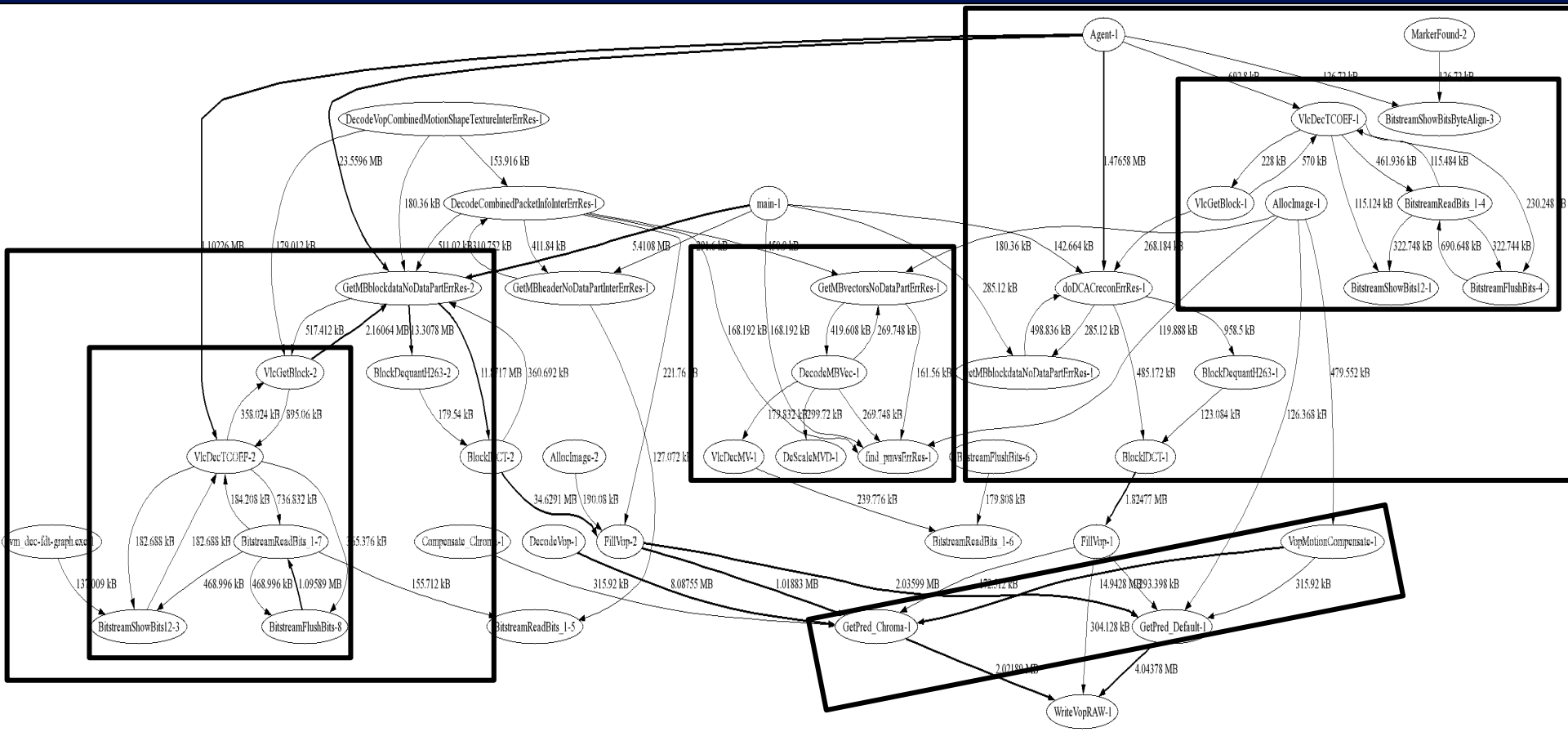
a) Function-call tree

b) Functions, functional modules and data-transfer dependences

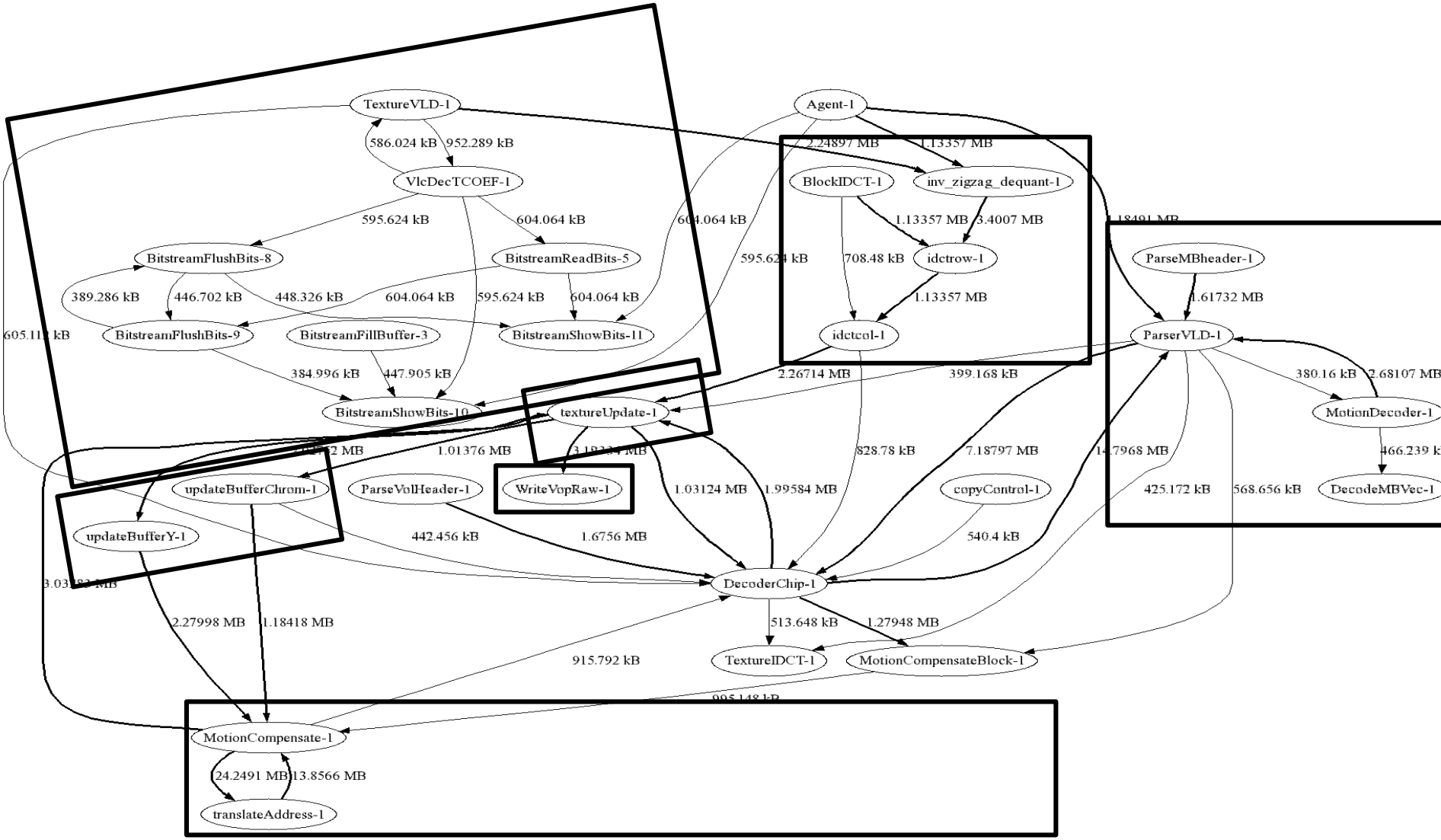




MPEG-4 Part 7 Optimized Reference Software



MPEG-4 video architectural C code



Conclusion

- Meaningful to analyze complexity at “algorithmic” level
- Appropriate metrics for the “Core experiments” aiming at developing low complexity should include and combine:
  - Operators and data type profiling
  - Memory size and accesses to a (virtual) memory architecture (local buffer or cache and external memory)
  - Critical path analysis (efficient optimized implementations)
  - Data flow architecture
- Using automatic tools developed for the exploration of the design space :
  - C much better than C++!!

- Thank you for your attention!!