www.thalesgroup.com

MINES ParisTech

A Complete Industrial Multi-target Graphical Tool-chain for Parallel Implementations of Signal and Image Applications

T. Petrisor, E. Lenormand, **C. Ancourt**
R. Barrere, M. Barreteau, F. Irigouin

ArtistDesign NoE, Map2MPSoC, 2011

THALES

Outline

Parallel programming in the embedded systems industry

A tooled approach for the industry

- **The Source-to-Source Compiler PIPS**
- **The multi-target tool-chain SpearDE**

Large embedded systems industry going parallel

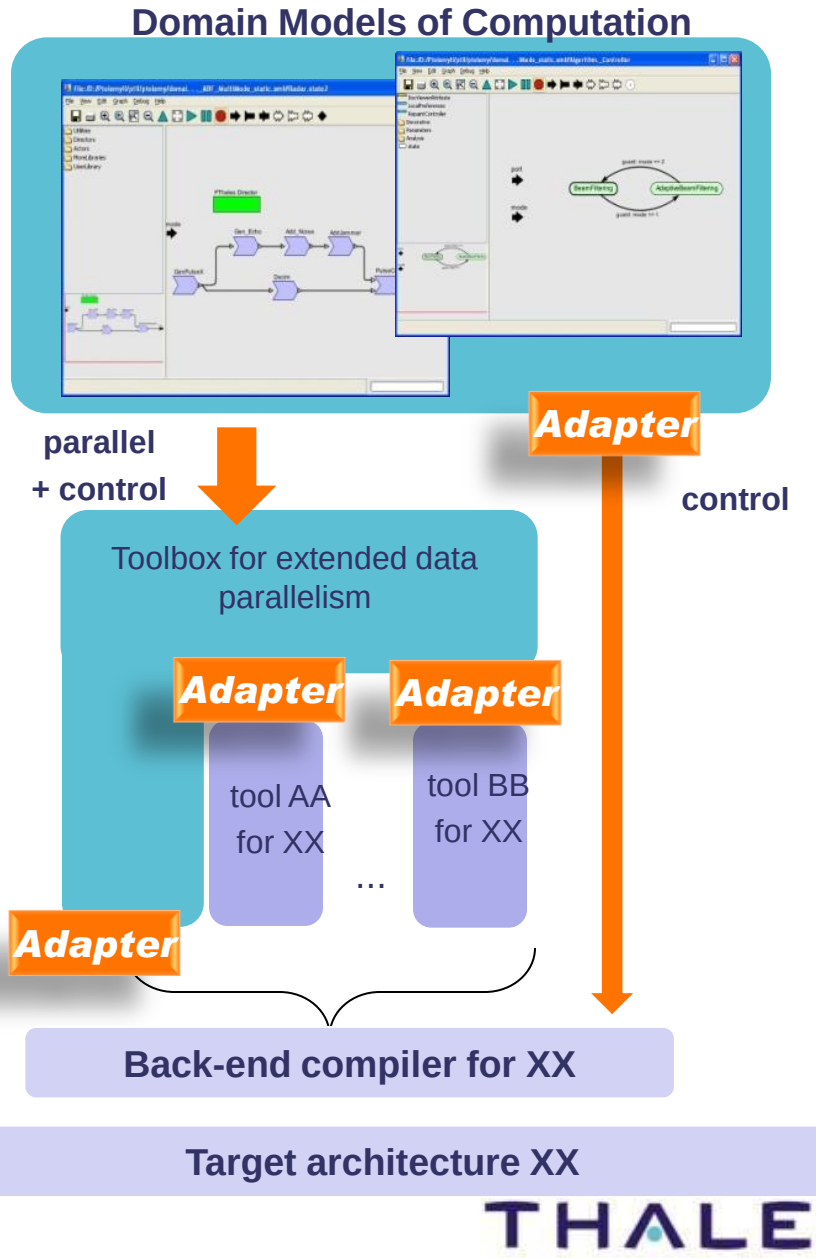
Main issues

- Shift from sequential to parallel by limiting loss in productivity
  - Make it possible for less expert users (engineers)
  - Lower development time
  - Traceability
- Address a variety of heterogeneous programmable parallel platforms

Domain specific approach

- User interface per domain is missing
  - Lack of mature tools
  - Lack of parallel programming languages consensus / parallel programming reserved for expert users
- Human in the loop is still needed

Applications: mixture of highly parallel parts and control parts



Interactive Design Workflow

System Level Specification

```
// Copyright THALES 2010 All rights
reserved
/* ------ Functions ----- */
void main_PE0(int dim1,int dim2,int dim3,
CplfIoat STIM[dim1][dim2][dim3], int
tab_index[5], int dimV1, int dimV2, CplfIoat
ValSteer[dimV1][dimV2], int dimR1, int
dimR2, int dimR3, CplfIoat
CI_out[dimR1][dimR2][dimR3] ){
 int i0,i1;
…
 CplfIoat mti_out[2000][64][18];
/*[2304000][1]*/
 …
 for (i0=0;i0<19;i0++)
  for (i1=0;i1<64;i1++)
   SEL(2000, X_2_out[i0][i1], tab_index,
40, sel_out[i0][i1]);
 for (i0=0;i0<19;i0++)
  COR(200,64,
sel_out[i0],sel_out[i0],CORR_out[i0]);
 …
}/* End of main */
```

PIPS4U

SpearDE

Architecture model

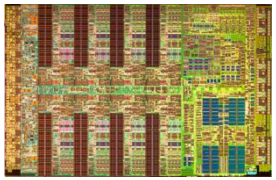
Application model

Generated code

Build from scratch

Backend Tools

©nVidia


©IBM

©TI

MINES ParisTech

THALES

Towards a parallelization toolchain for the industry

PIPS – a source-to-source compiler

PIPS4U

- Input: Fortran and C applications

- Semantic Analyses
- Loop transformations
- Program transformations
- Pretty printers
- Source-to-source code generation

Semantic Analyses :

- **Effects Read/Write**
- **Transformers**
- **Preconditions**
- **Array Regions**
- **Effects IN/OUT**
- **Dependences**
- **Complexity**

Loop Transformations:

- Distribution
- Interchange
- Strip Mining
- Tiling
- Unrolling
- Fusion
- Allen & Kennedy Parallelization
- Coarse Grain Parallelization

Program Transformations:

- Array Expansion
- **Privatization**
- Scalarization
- Cloning
- Common Subexpression Elimination
- Dead Code Elimination
- **Constant Propagation**
- Forward Substitution
- Invariant Code Motion
- Inlining
- Outlining
- USE/DEF Elimination
- **Control Simplification**

Prettyprinter :

- Source code C, Fortran 77 with directives
- Source code with analysis results
- XML
- Call tree, call graph
- Interprocedural control flow graph

Code Generation :

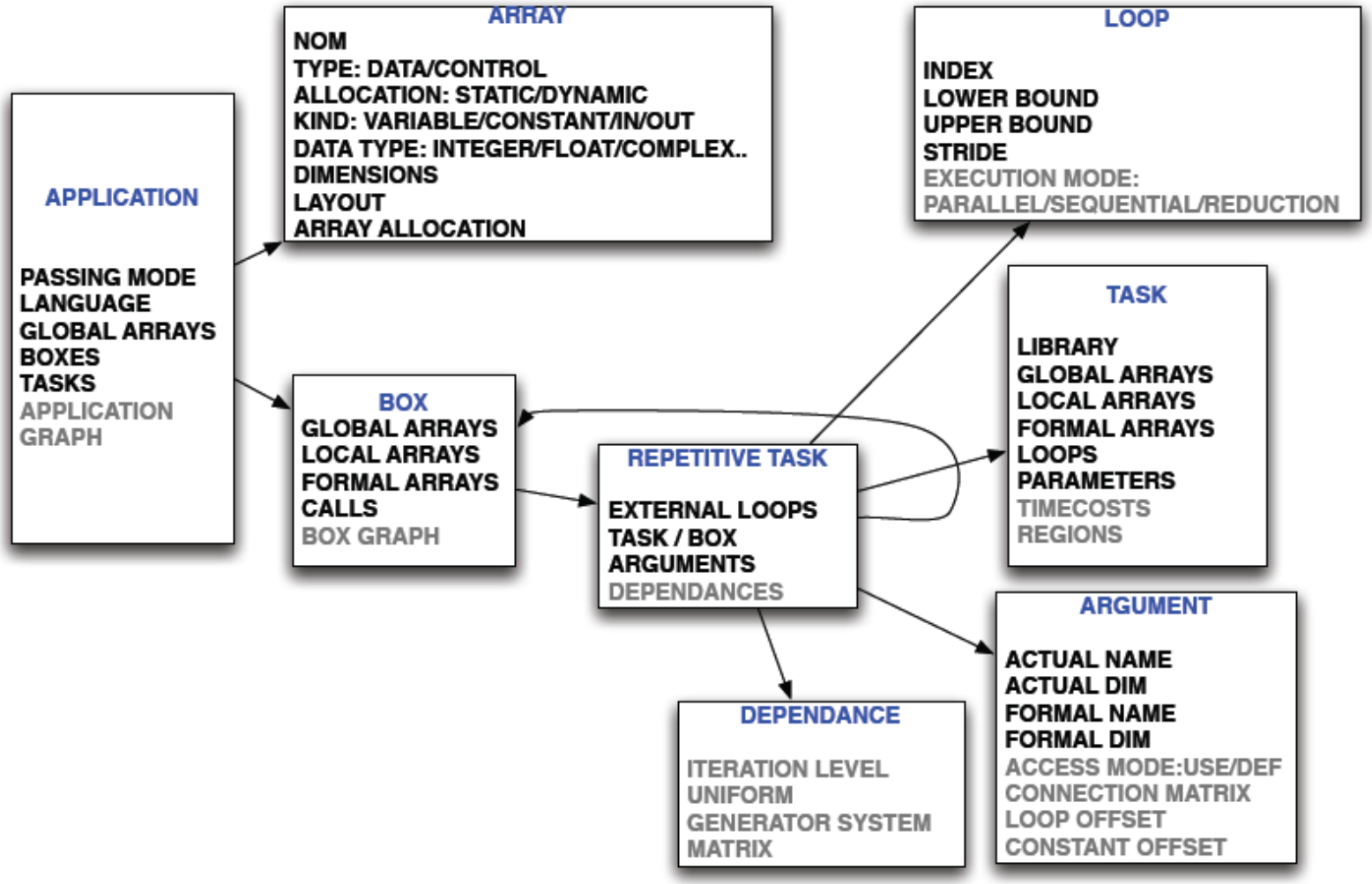
- OpenMP
- MPI
- SSE/AVX/NEON
- GPU/CUDA
- Terapix

MINES ParisTech

THALES

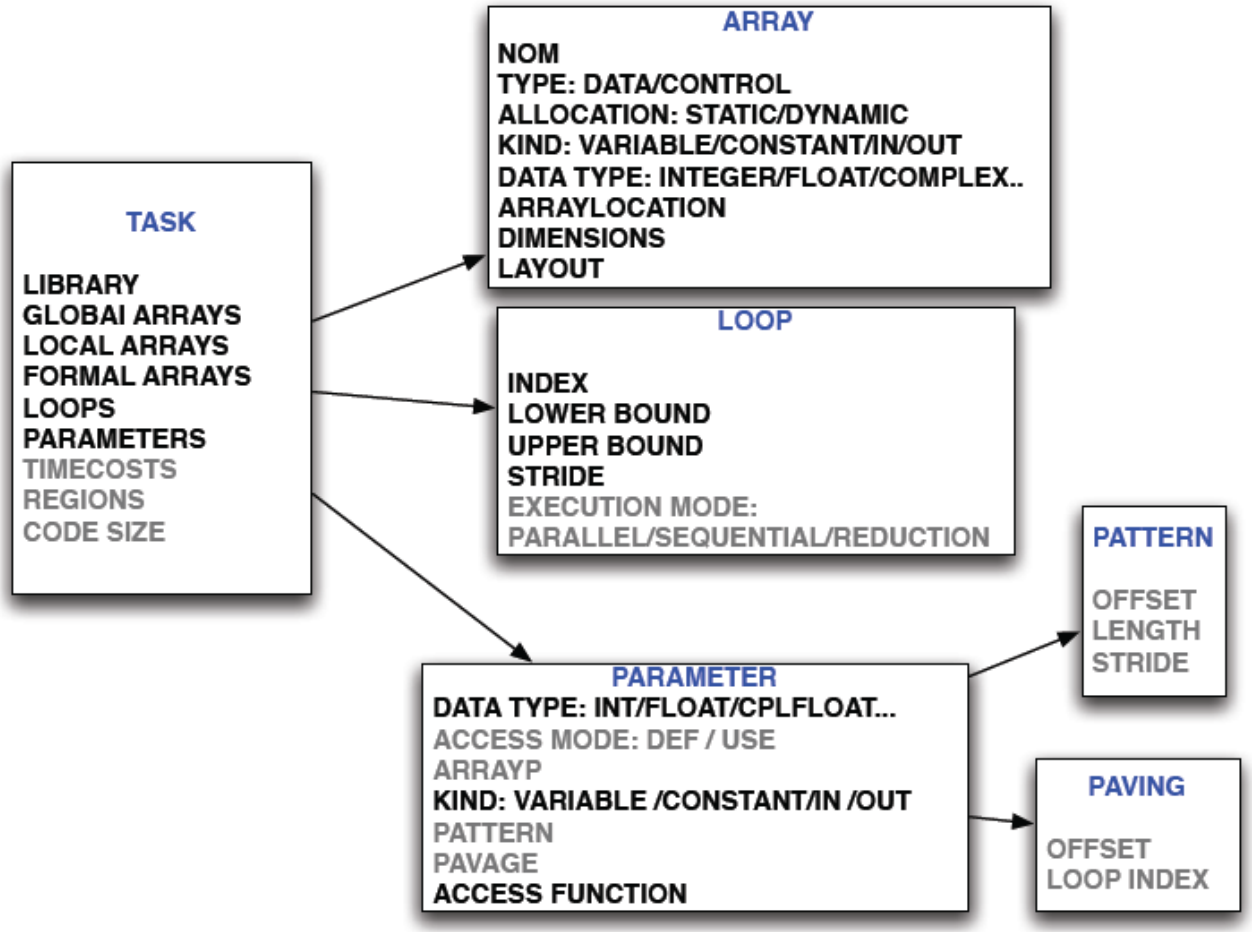
Modeling with PIPS (1)

Application model



Modeling with PIPS (2)

Task model



Coding rules for parallelization

Goals

- Automatic modeling of applications and library functions
- Automatic optimization of tasks

Rules

- No recursive function
- Use precise typing and proper array declaration
- Avoid pointer arithmetic and linearized array

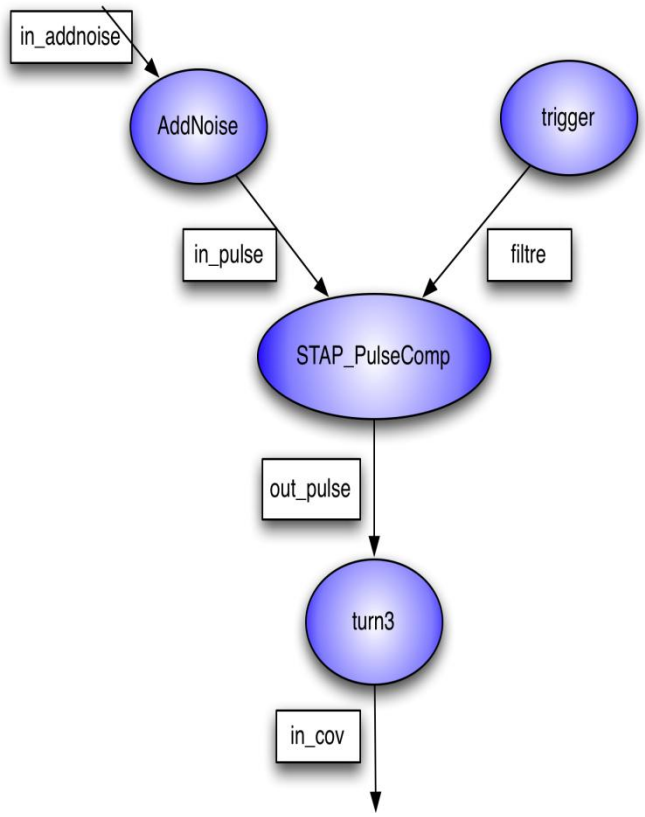
A repetitive task call example

```
<Call Name="STAP_PulseComp">
<ExternalLoops>
    <Loop Index="j" ExecutionMode="PARALLEL">
    <LowerBound>
        <Symbolic>0</Symbolic>
        <Numeric>0</Numeric>
    </LowerBound>
    <UpperBound>
        <Symbolic>nsa-1</Symbolic>
        <Numeric>4</Numeric>
    </UpperBound>
    <Stride>
        <Symbolic>1</Symbolic>
        <Numeric>1</Numeric>
    </Stride>
    </Loop>
    <Loop Index="k" ExecutionMode="PARALLEL">
    <LowerBound>
        <Symbolic>0</Symbolic>
        <Numeric>0</Numeric>
    </LowerBound>
    <UpperBound>
        <Symbolic>nrec-1</Symbolic>
        <Numeric>31</Numeric>
    </UpperBound>
```

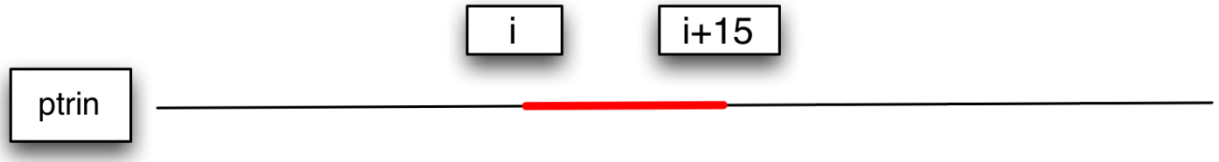
```
for(j=0;j<nsa;j++)

    for(k=0;k<nrec;k++)

        STAP_PulseComp(tv,
            in_pulse[j][k],
            tf,filtre, out_pulse[j][k]);
```

Task Graph

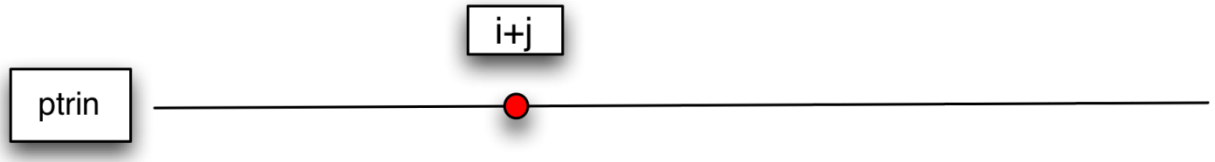
```
<BoxGraph Name="trt_burst">
    ....
    <TaskRef Name="AddNoise">
      <Computes ArrayName="in_pulse"/>
      <Needs ArrayName="in_addnoise"
    DefinedBy="Echo_Raf"/>
    </TaskRef>
    <TaskRef Name="STAP_PulseComp">
      <Computes ArrayName="out_pulse"/>
      <Needs ArrayName="filtre"
    DefinedBy="trigger"/>
      <Needs ArrayName="in_pulse"
    DefinedBy="AddNoise"/>
    </TaskRef>
    <TaskRef Name="turn3">
      <Computes ArrayName="in_cov"/>
      <Needs ArrayName="out_pulse"
    DefinedBy="STAP_PulseComp"/>
    </TaskRef>
    ........
</BoxGraph>
```



Convex Array Regions (1)




```
//  <ptrin[PHI1].im-R-EXACT-{0<=PHI1, PHI1+tf<=tv+15, tf==16, tv==95}>
///  <ptrin[PHI1].im-R-EXACT-{i<=PHI1, PHI1<=i+15, tf==16, tv==95, 0<=i, i<=79}>
//  <ptrin[PHI1].re-R-EXACT-{i<=PHI1, PHI1<=i+15, tf==16, tv==95,0<=i, i<=79}>
     for(j = 0; j <= tf-1; j += 1) {
```




```
//  <ptrin[PHI1].im-R-EXACT-{i+j==PHI1, tf==16, tv==95, 0<=i, i<=79,0<=j, j<=15}>
//  <ptrin[PHI1].re-R-EXACT-{i+j==PHI1, tf==16, tv==95, 0<=i, i<=79,  0<=j, j<=15}>
        R += ptrin[i+j].re*ptrfiltre[j].re-ptrin[i+j].im*ptrfiltre[j].im;
//  <ptrin[PHI1].im-R-EXACT-{i+j==PHI1, tf==16, tv==95, 0<=i, i<=79, 0<=j, j<=15}>
//  <ptrin[PHI1].re-R-EXACT-{i+j==PHI1, tf==16, tv==95, 0<=i, i<=79, 0<=j, j<=15}>
        I += ptrin[i+j].im*ptrfiltre[j].re+ptrin[i+j].re*ptrfiltre[j].im;
    }
```

Convex Array Regions (2)


```
// <ptrin[PHI1].im-R-EXACT-{0<=PHI1, PHI1+tf<=tv+15, tf==16, tv==95}>
// <ptrin[PHI1].re-R-EXACT-{0<=PHI1, PHI1+tf<=tv+15, tf==16, tv==95}>
void STAP_PulseComp(int tv, CpIfloat ptrin[tv], int tf, CpIfloat ptrfiltre[tf],
                                    CpIfloat ptrout[tv-tf+1]) {
// <ptrin[PHI1].im-R-EXACT-{0<=PHI1, PHI1+tf<=tv+15, tf==16, tv==95}>
// <ptrin[PHI1].re-R-EXACT-{0<=PHI1, PHI1+tf<=tv+15, tf==16, tv==95}>
  for(i = 0; i <= tv-tf; i += 1) {
    R = 0.0;
    I = 0.0;
   for(j = 0; j <= tf-1; j += 1) {
        R += ptrin[i+j].re*ptrfiltre[j].re-ptrin[i+j].im*ptrfiltre[j].im;
        I  += ptrin[i+j].im*ptrfiltre[j].re+ptrin[i+j].re*ptrfiltre[j].im;
   }
  …
```

Linear Algebra Based Formalism

Actual declaration f $0 \leq F^{-1}\ f \leq 1\ ou\ 0 \leq f \leq F$

Layout of array r $a = l \cdot r + a_0$

Actual array r $r = \Phi = \Omega^A_{fit}\ m + \Omega^A_{pav}\ i + k^A$

Formal array f $f = \Phi = \Omega^{TE}_{fit}\ m + \Omega^{TE}_{pav}\ \ i^{TE} + k^{TE}$

Call site: $r = C\ f + c$

Mapping of iterations: $i = P\ i^A + Q\ i^{TE}$

Interface constraint: $C\ \Omega^{TE}_{pav} = \Omega^A_{pav}\ Q$

PIPS contributions

- Fortran and C code -> Application and library function models
- Parallelism and reduction detection
- *Patterns* of array elements referenced by task
- Predicates on variables: constant propagation
- Convex Array Regions: coarse-grain parallelization, memory size estimation
- Data flow graph
- Time estimation

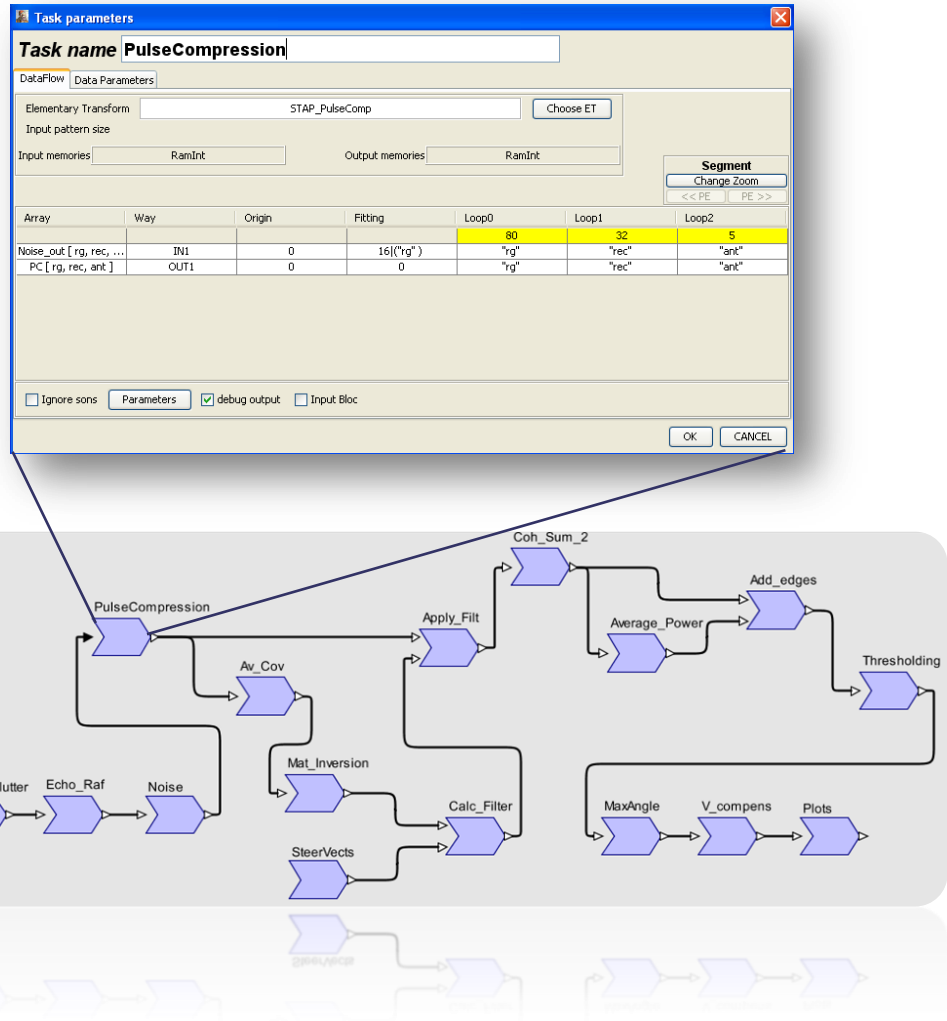
SpearDE flow: Application model

Task and data parallelism made explicit

- Graphical interface
  - Information is structured!

Methodology

- Multi-dimensional Synchronous Dataflow graph
- Array-OL formalism
- Library of elementary operators (application agnostic)



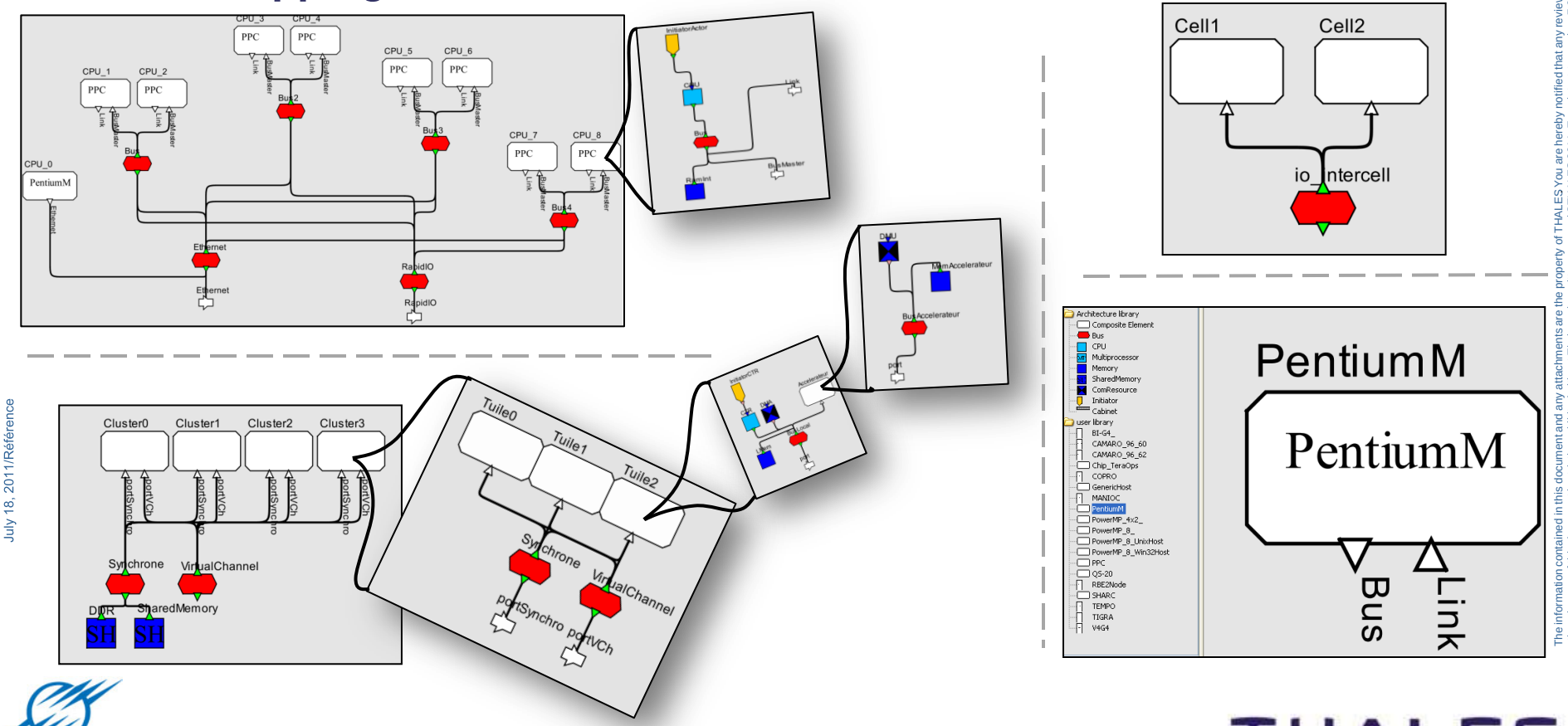
MINES ParisTech

THALES

SpearDE flow: Architecture model

Support heterogeneous architectures

- No underlying topology assumptions
- Abstract representation highlighting only the basic blocks involved in mapping: CPUs, local/shared memories, buses

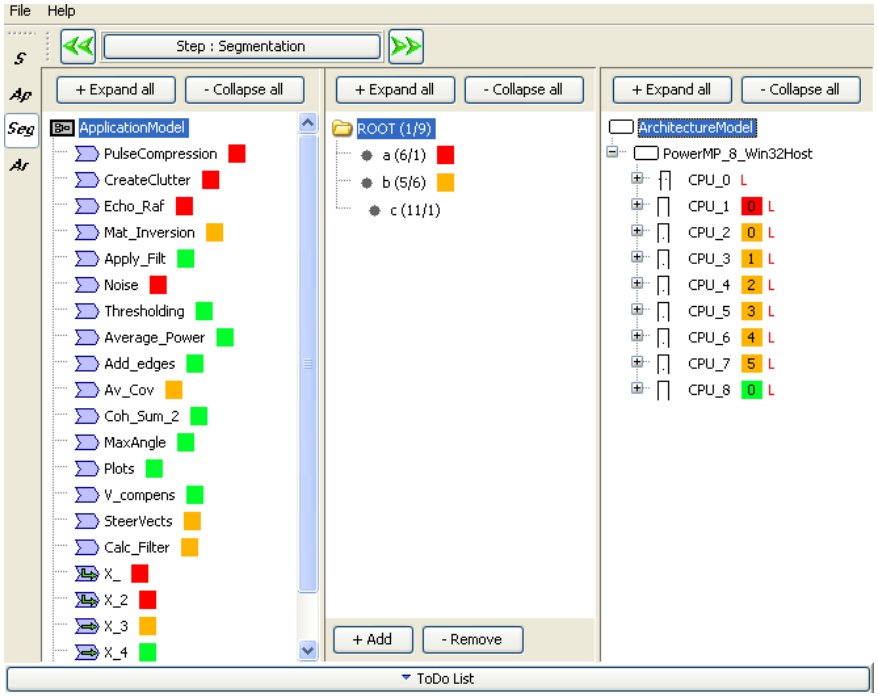


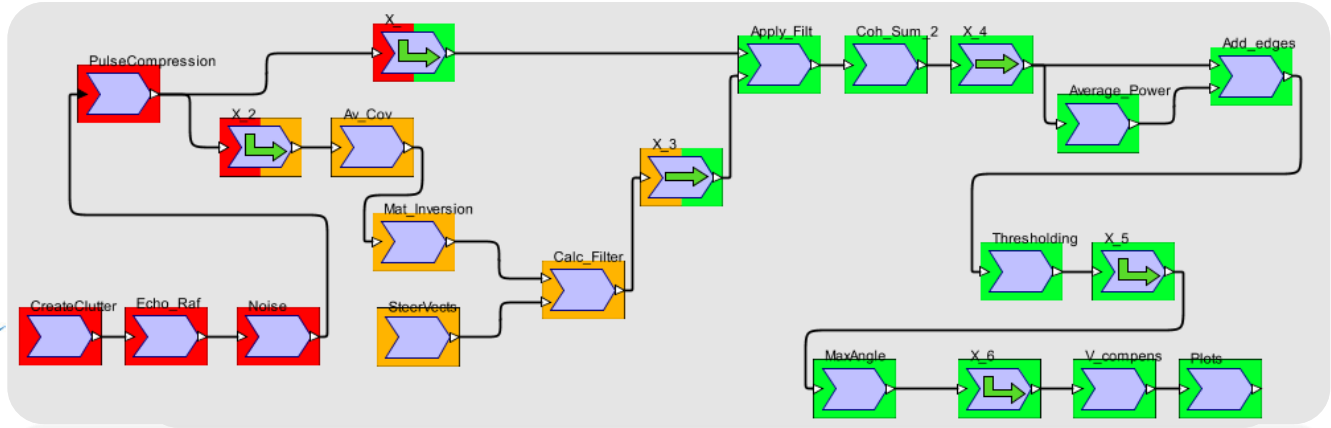
MINES ParisTech

THALES

Application to Architecture Mapping

- Keep it manual as long as general optimization heuristics/criteria unknown
- Rapid prototyping
- Tracing mapping decisions





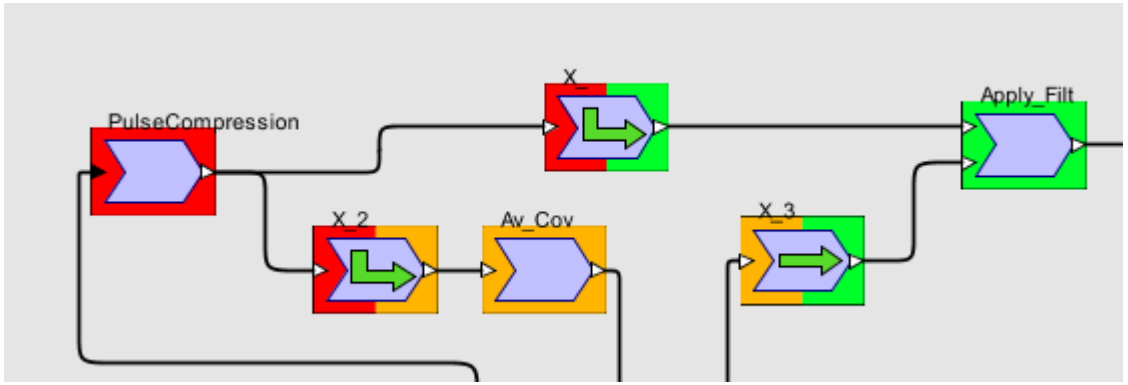
MINES ParisTech

THALES

Communications

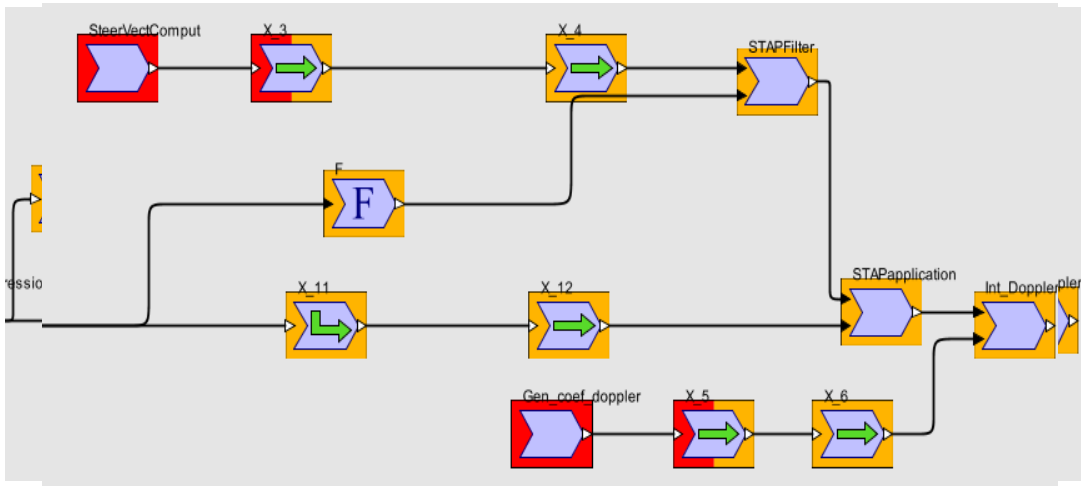
Legal data transfers

Data reorganization



- Memory estimation

Fusion



Static scheduling enabling code generation

MINES ParisTech

THALES

Code generation

Various code generators

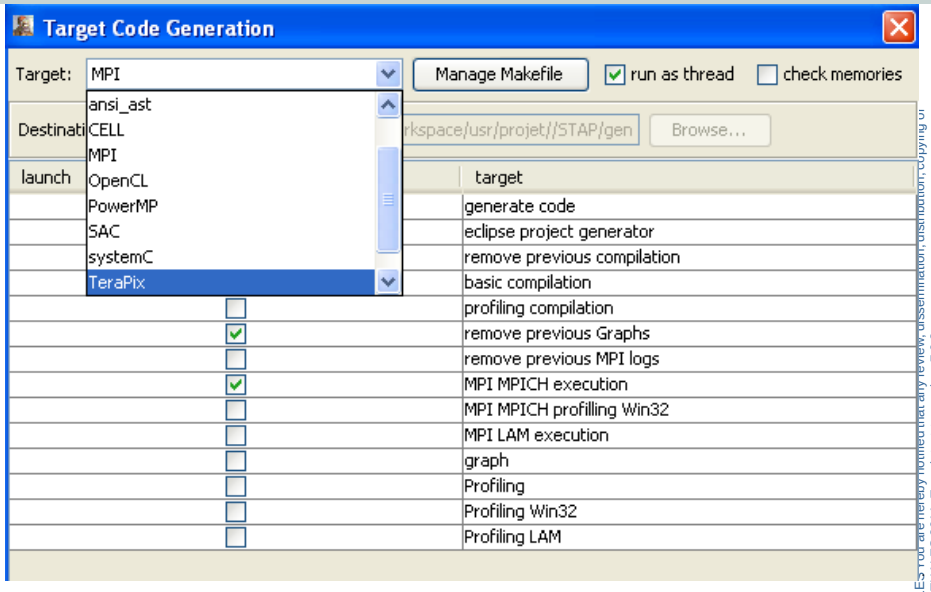
- Sequential C for functional debug
- MPI
- SystemC
- OpenCL/GPU
- Custom in-house FPGA architecture (Terapix, …)

Glue code takes up a lot of space as compared to kernel code

- Error-prone and difficult to debug

Performance analysis

Back-end tools can then optimize the kernel code for the chosen platforms




```
int i__ApplicationModel_F_0 = 0;
int i__ApplicationModel_F_1 = 0;
int N_ApplicationModel_F = 0;
int i__ApplicationModel_F2_0 = 0;
int i__ApplicationModel_F2_1 = 0;
int N_ApplicationModel_F2 = 0;
int idxTime = 0;
int firstBloc = 0;

MPI_Request req_ApplicationModel_X__out[8];
MPI_Request req_ApplicationModel_X_2_out[24];

/* ------ Memory Mapping ----- */
typedef union global {
struct Sega{
/* S1 declaration */
struct S1 {
  Cplfloat _ApplicationModel_Gen_chirp_out[65536]; /*[65536][1]*/
  Cplfloat _ApplicationModel_Task2_out[49408]; /*[49408][1]*/
}S1;
}Sega;
…
void main_PE0(){
…
```


MINES ParisTech

THALES

Summary

- **The embedded industry is driven by different criteria when it comes to shifting to parallel architectures**
- **Strong need for tools bridging the gap between system level design and hardware implementation**
  - With minimum extra costs (time, expertise…)
  - By keeping the man in the loop as long as automation does not yield a result 100% of the time
- **Toolflow enabling the appropriate abstractions at different levels in the design and implementation process**
  - Applications using models of computation exhibit parallelism vs. non-parallelism properties
  - Input models may be built from scratch (new applications) or with source-to-source tools (PIPS)
  - Abstractions of architectures allow rapid mapping
  - Functional verification
  - Mapping decisions tracking
  - Performance evaluation