A complete industrial multi-target graphical tool-chain for parallel implementations of signal/image applications

Teodora Petrisor, Eric Lenormand
Thales Research and Technologies/LSE
1 Av. Augustin Fresnel, 91767 Palaiseau, France
{claudia-teodora.petrisor, eric.lenormand}@thalesgroup.com

Corinne Ancourt, François Irigoin
MINES ParisTech/CRI
35 rue Saint-Honoré, 77305 Fontainebleau, France
{corinne.ancourt, francois.irigoin}@mines-paristech.fr

Abstract: Today, applications developed in an industrial environment need to shift from a sequential implementation to one or several different parallel ones, under stringent productivity and performance constraints. As this operation is generally complex and moreover needs particular skills which are not part of the traditional software engineering cultural background, there is a clear need for tools to assist it.

A parallel programming tool-chain must be able to address various parallel target hardware platforms. This may be due to obsolescence of components, to different environmental constraints, or different requirements of applications that make some candidate targets more efficient than others. So the tool-chain must be easily adaptable to heterogeneous compositions of FPGAs, general purpose processors, DSPs, GPUs, etc. while hiding as much as possible the cumbersome details of these architectures to the user. However, with the current state of the art, except for very simple cases, it seems difficult to find sufficiently robust completely automated tool-chains (tackling the often tricky architecture-specific trade-offs done for mapping), especially when real time and performance are needed. Thus, a man-in-the-loop approach is preferred, where the user can experiment different variants through fast prototyping, performance analysis, and rapid design space exploration. This implies providing the user with sufficiently abstracted views of both application and architecture, giving visibility on the information that is relevant to parallelization and mapping, as well as the commands enabling him/her to drive the main mapping decisions.

We present here such models of both applications and architectures, and two complementary tools handling them and covering the entire design flow. The first tool based on the source-to-source compiler PIPS[1], generates application models from application code, and the second tool, SpearDE[2], supports the flow from models to code generation. We address the signal and image processing domains.

Application models expose potentially parallelizable parts, consistently with control and/or non parallelizable parts. The modeling process implemented in PIPS creates them automatically from an annotated C code. This may be a – generally re-factored – version of legacy code. Pieces of code that are subject to mapping are more detailed in the models. They are presented as dataflow graphs, where edges support multi-dimensional arrays of data and nodes are nested loops iterating computational kernels, e.g. from a library. The analysis provides in particular information on arrays and on data dependencies both at graph level (i.e. between nodes) and at kernel level, as well as estimations of the execution times.

The application data-flow graphs are then handled by SpearDE, which also holds the architecture models which provide a structural view (processing units, memory layout, communication paths) as well as a performance view (such as time behavioral models of the architecture elements) of the architecture.

Based on these two models a mapping strategy can be defined, addressing both task-level and data-level parallelism. The tedious operations linked to mapping (which are also error prone when done manually), such as data transfers, data reordering or model consistency in general are handled by the tool. These are intrinsically linked to the target hardware and different for each configuration regardless the fact that the application model might be the same. Next, the tool generates a valid scheduling, including software pipelining for the mapped application, as well as the involved static memory layout. This leads to rapid performance simulation on the given platform and therefore allows the user to iteratively choose the best design according to the imposed constraints. It is worth noting that this user-driven iterative process can also contribute to the flexibility of the tool because performance constraints might vary for the same application, depending on the equipment context.

Finally, code generation is performed using back-end, processor-specific compilers. Mapping/code generation with SpearDE has been experimented on multi PowerPC and multi-DSPs architectures, IP-based processing chains on FPGAs, GPUs, Cell, and in-house SIMD architectures on FPGA.

[1] F. Irigoin, P. Jouvelot and R. Triolet – "*Semantical interprocedural parallelization: An overview of the PIPS project*", in International Conf. On Supercomputing, Cologne, June 1991

[2] E. Lenormand and G. Edelin – "*An industrial perspective: A pragmatic high-end signal processing design environment at Thales*", in SAMOS-III, Computer Systems: Architecture, Modeling and Simulation, LNCS 3133, Springer, September 2003

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

Research & Technology

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

MINES ParisTech

THALES

Large embedded systems industry going parallel

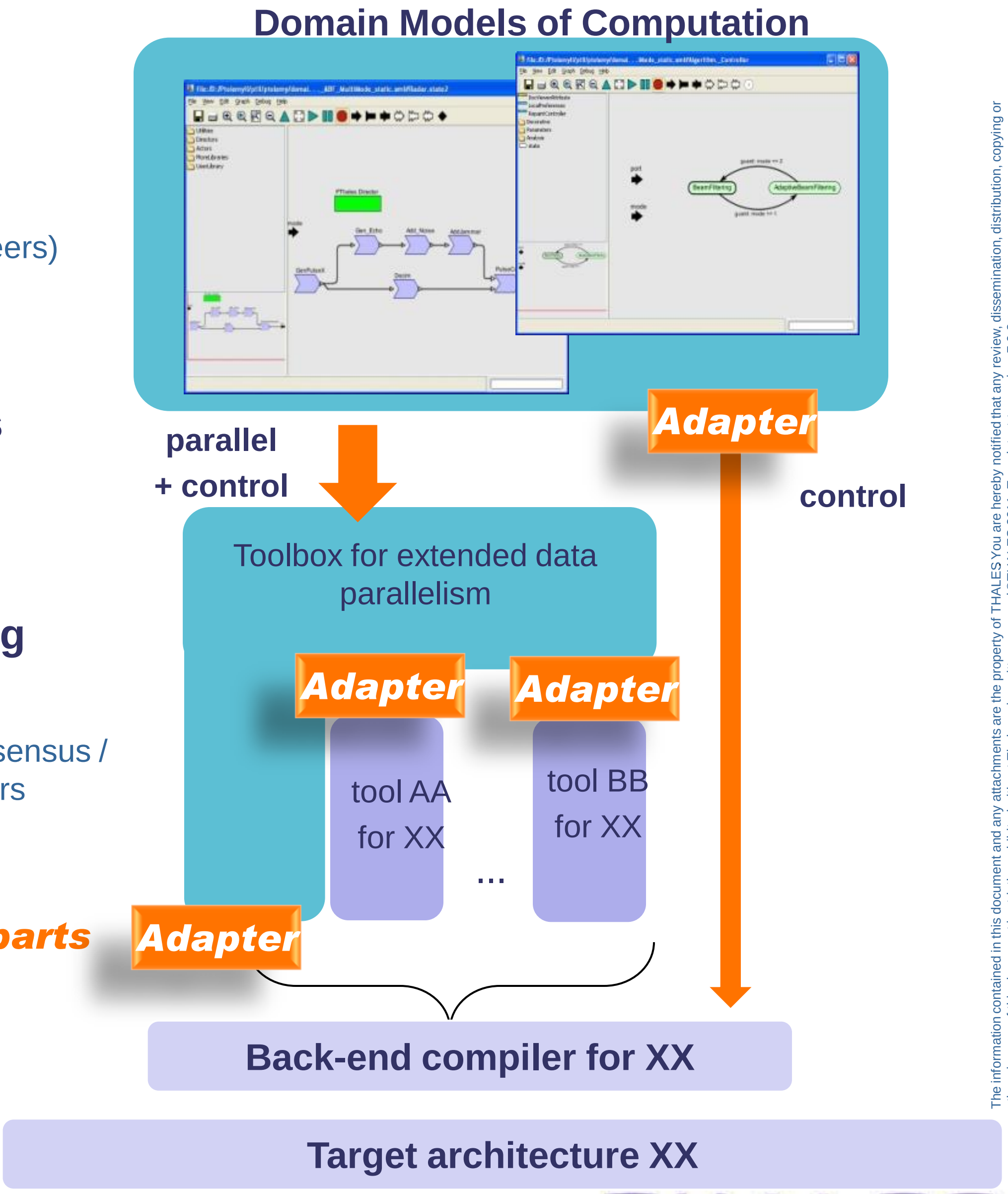
Main issues

- **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**

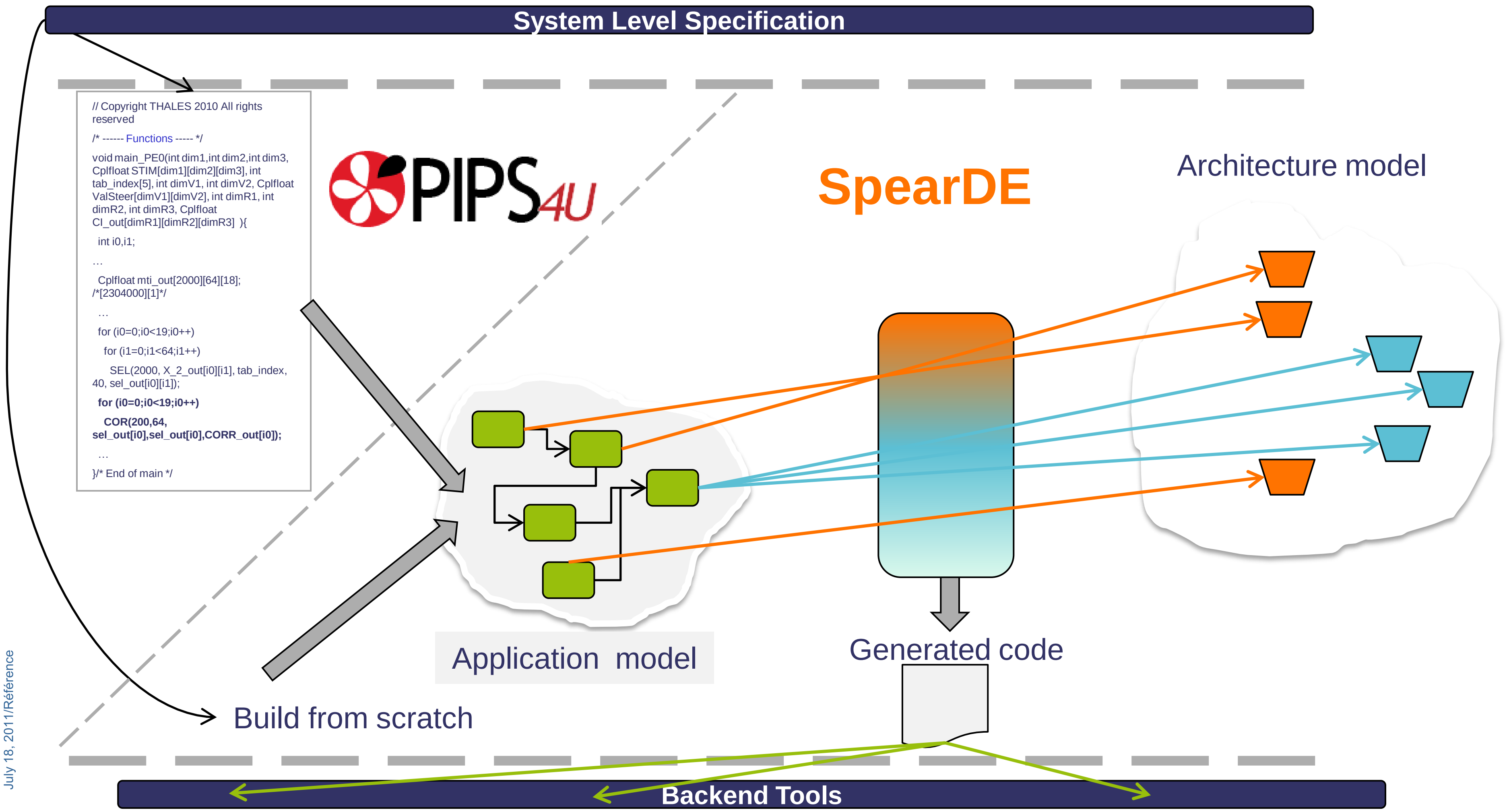
Domain specific approach

- **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

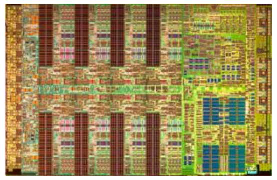


Interactive Design Workflow



```
// Copyright THALES 2010 All rights
reserved
/* ------ Functions ----- */
void main_PE0(int dim1,int dim2,int dim3,
Cplfloat STIM[dim1][dim2][dim3], int
tab_index[5], int dimV1, int dimV2, Cplfloat
ValSteer[dimV1][dimV2], int dimR1, int
dimR2, int dimR3, Cplfloat
CI_out[dimR1][dimR2][dimR3]  ){
  int i0,i1;
…
  Cplfloat 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 */
```

©nVidia


©IBM

Scalable, high-performance, fixed-point
digital signal processing
©TI

MINES ParisTech

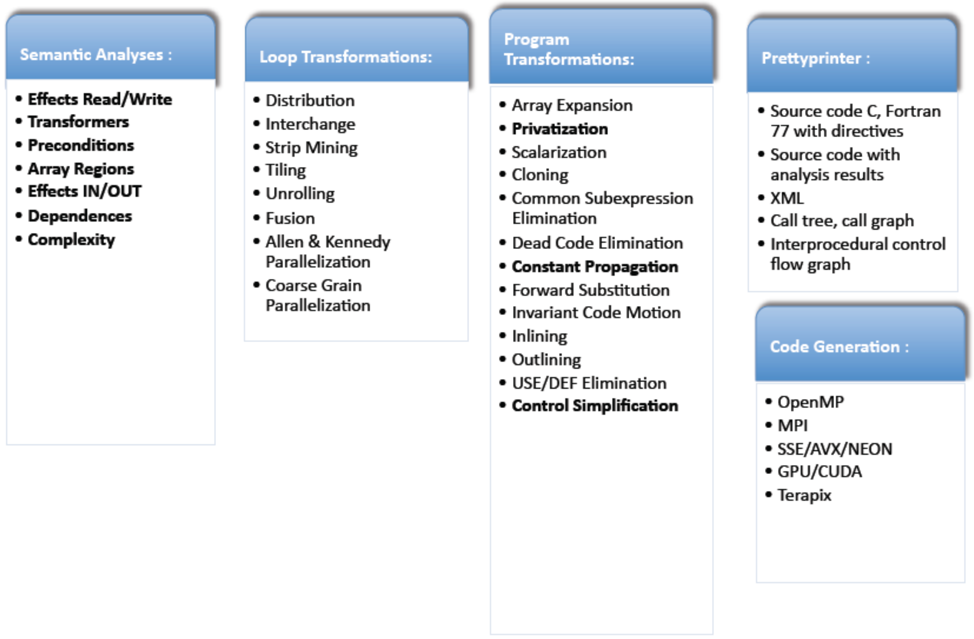
THALES

Towards a parallelization toolchain for the industry

PIPS – a source-to-source compiler

- 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

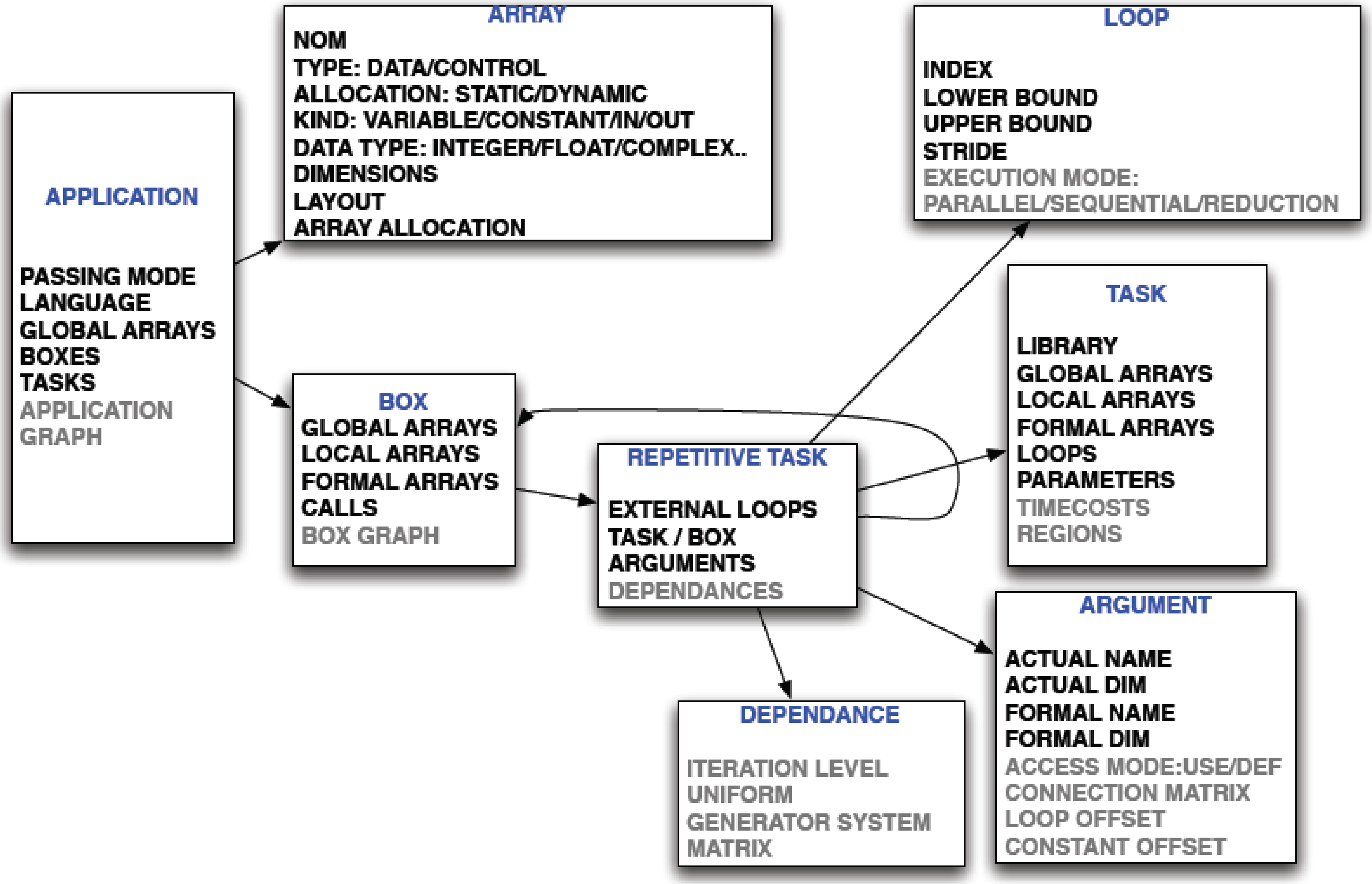
Program Transformations:
- Array Expansion
- **Privatization**
- Scalarization
- Cloning
- Common Subexpression Elimination
- Dead Code Elimination
- **Constant Propagation**
- Forward Substitution
- Invariant Code Motion
- Inlining
- Outlining
- USE/DEF Elimination
- **Control Simplification**

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

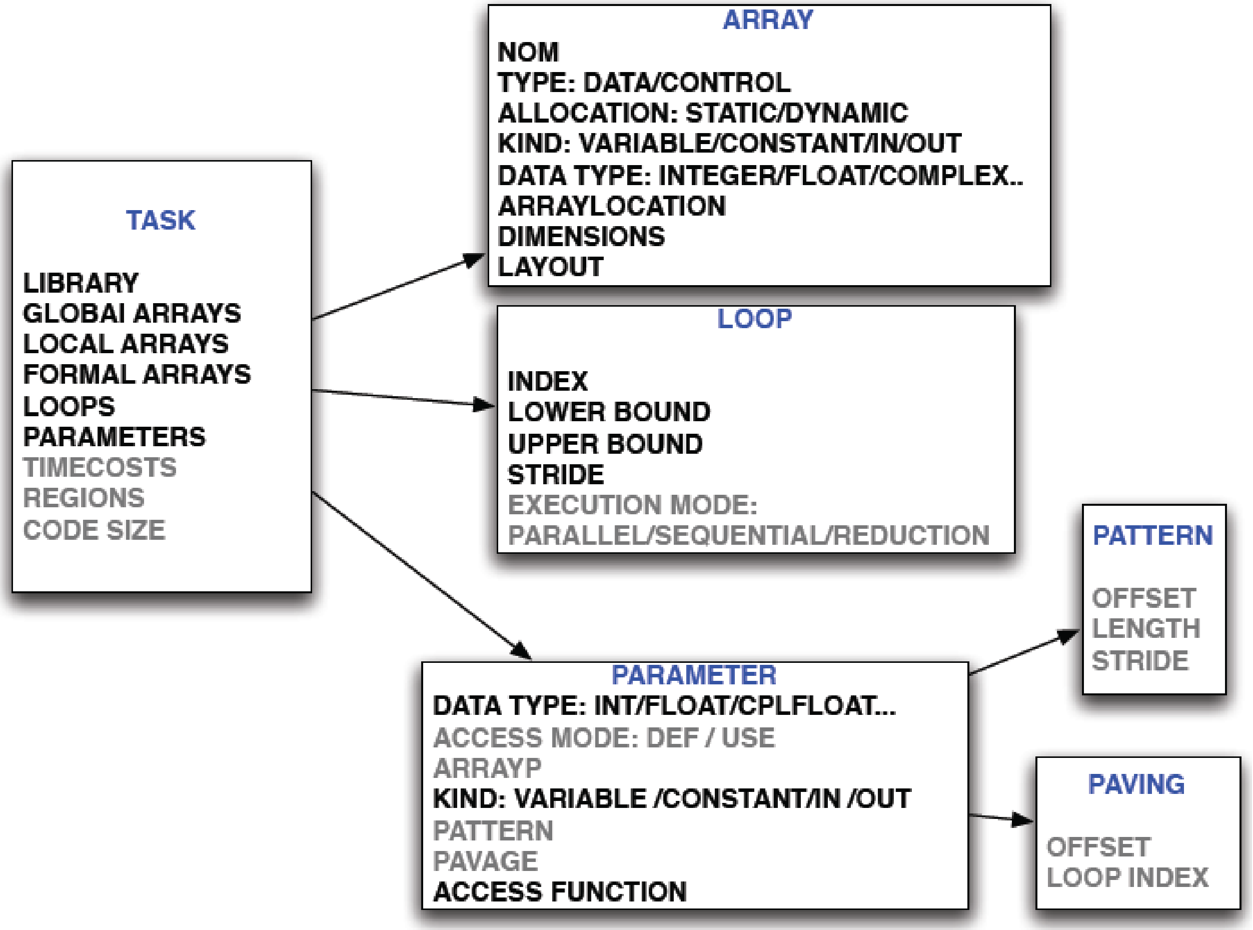
Modeling with PIPS (1)

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>
```

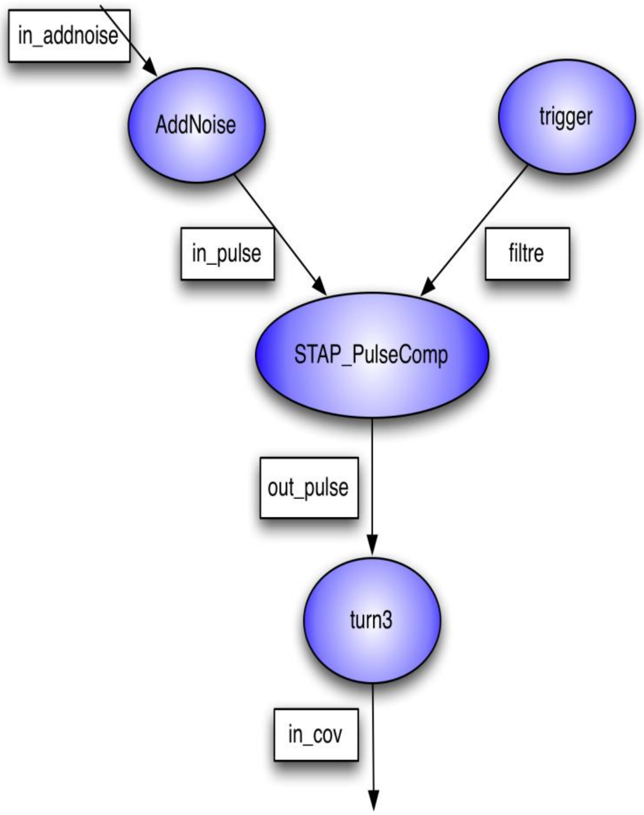
```
for(j=0;j<nsa;j++)

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

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

MINES ParisTech

THALES

Task Graph

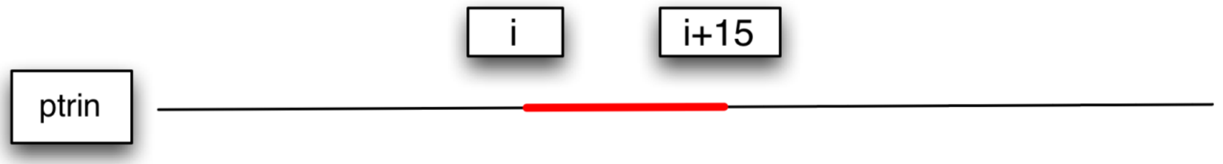
```
<BoxGraph Name="trt_burst">
      ....
    <TaskRef Name="AddNoise">
      <Computes ArrayName="in_pulse"/>
      <Needs ArrayName="in_addnoise"
    DefinedBy="Echo_Raf"/>
    </TaskRef>
    <TaskRef Name="STAP_PulseComp">
      <Computes ArrayName="out_pulse"/>
      <Needs ArrayName="filtre"
 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



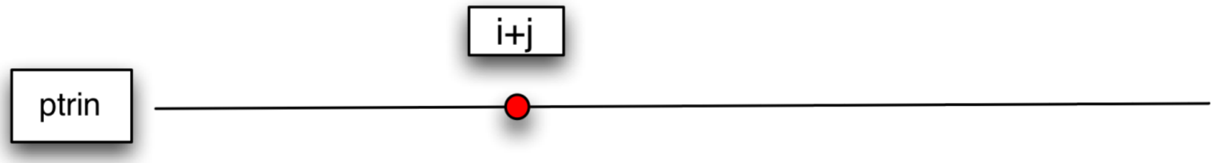
MINES ParisTech

THALES

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) {
```

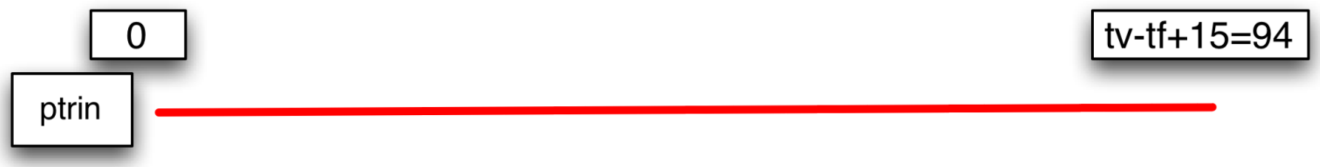


```
//  <ptrin[PHI1].im-R-EXACT-{i+j==PHI1, tf==16, tv==95, 0<=i, i<=79,0<=j, j<=15}>
//  <ptrin[PHI1].re-R-EXACT-{i+j==PHI1, tf==16, tv==95, 0<=i, i<=79,  0<=j, j<=15}>
      R += ptrin[i+j].re*ptrfiltre[j].re-ptrin[i+j].im*ptrfiltre[j].im;
//  <ptrin[PHI1].im-R-EXACT-{i+j==PHI1, tf==16, tv==95, 0<=i,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

MINES ParisTech

THALES

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, Cplfloat ptrin[tv], int tf, Cplfloat ptrfiltre[tf],
                                   Cplfloat 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;
   }
  …
```

MINES ParisTech

THALES

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

MINES ParisTech

THALES

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

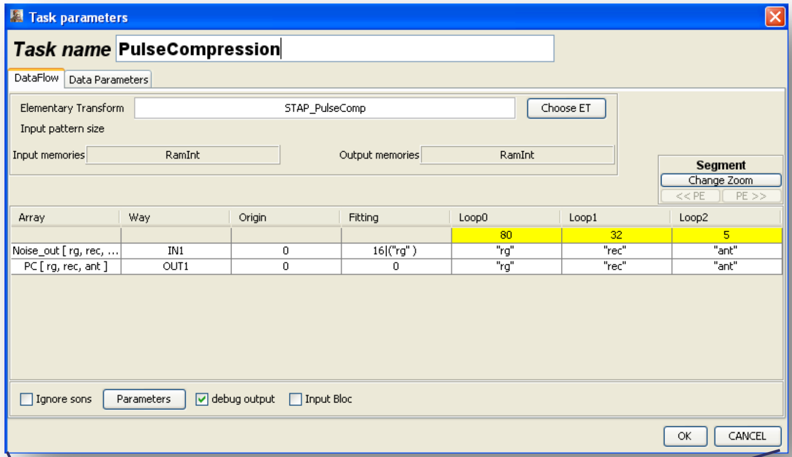
SpearDE flow: Application model

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)



Task parameters

Task name PulseCompression

DataFlow | Data Parameters

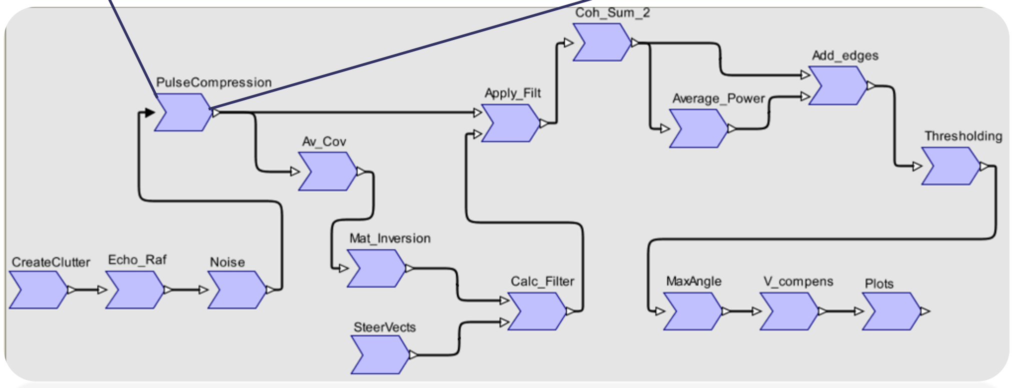
Elementary Transform: STAP_PulseComp

Input pattern size

Input memories: RamInt — Output memories: RamInt

Array	Way	Origin	Fitting	Loop0	Loop1	Loop2
				80	32	5
Noise_out [rg, rec, …	IN1	0	16|("rg")	"rg"	"rec"	"ant"
PC [rg, rec, ant]	OUT1	0	0	"rg"	"rec"	"ant"

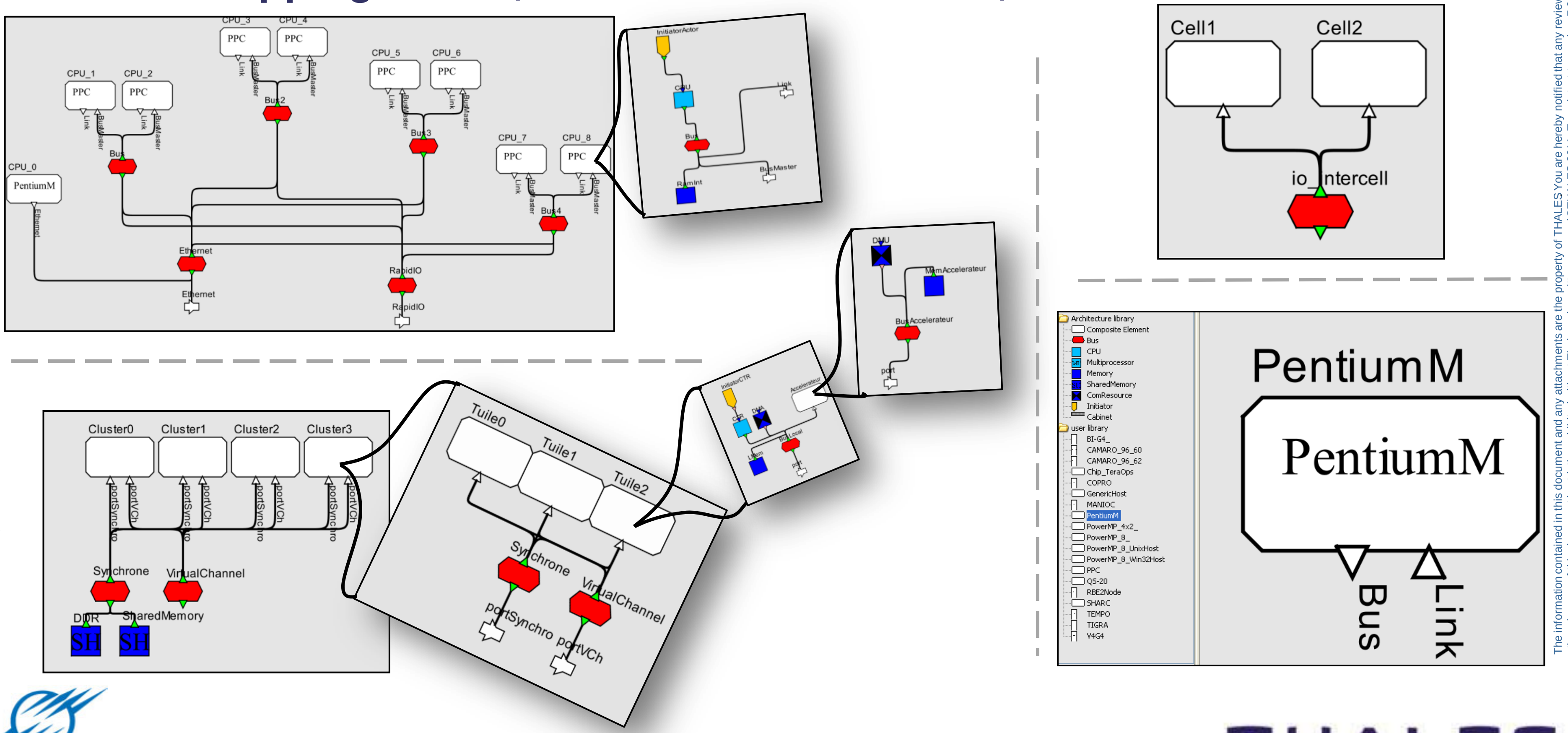
☐ Ignore sons — ☑ debug output — ☐ Input Bloc



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

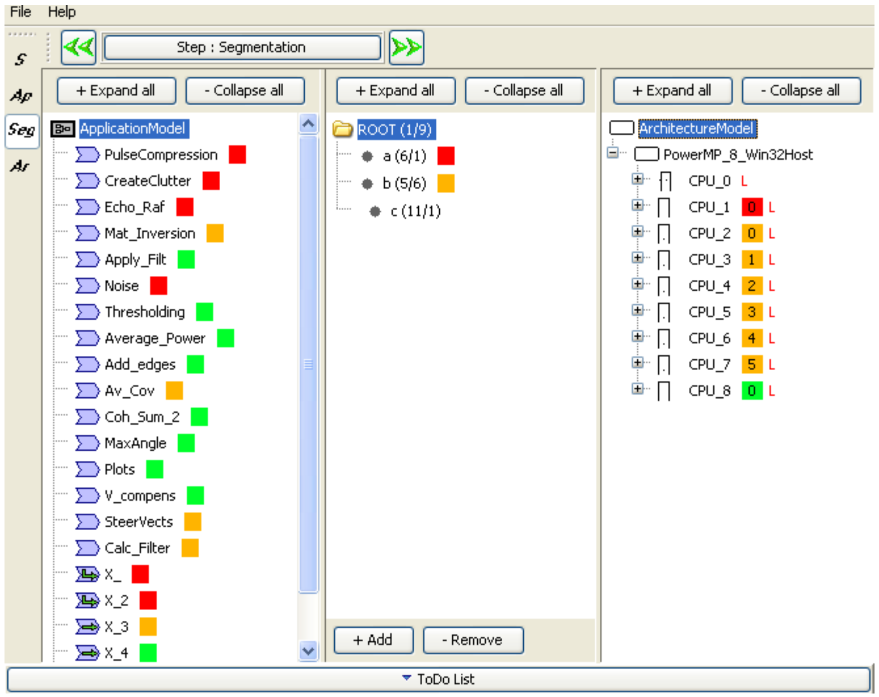


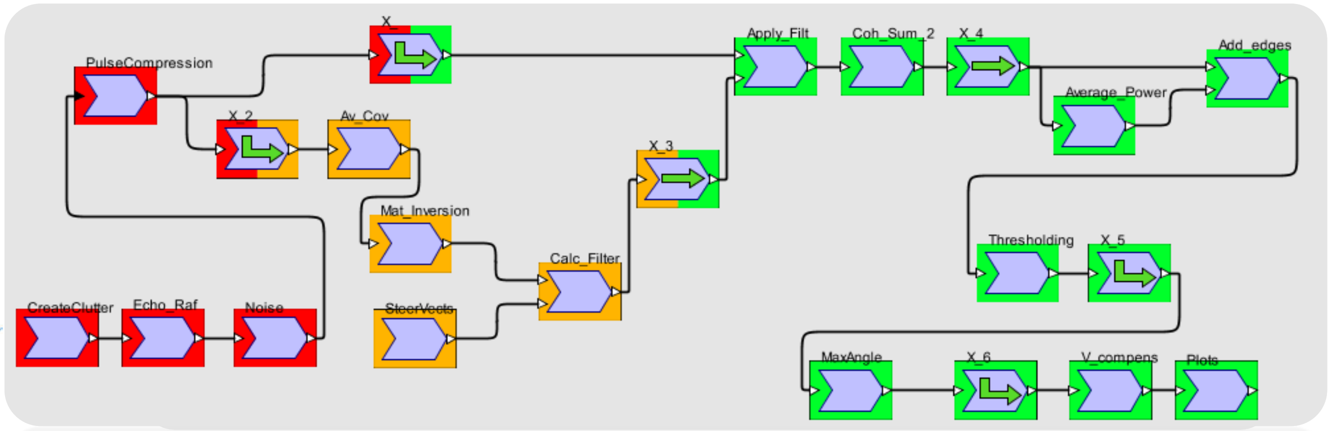
MINES ParisTech

THALES

Application to Architecture Mapping

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

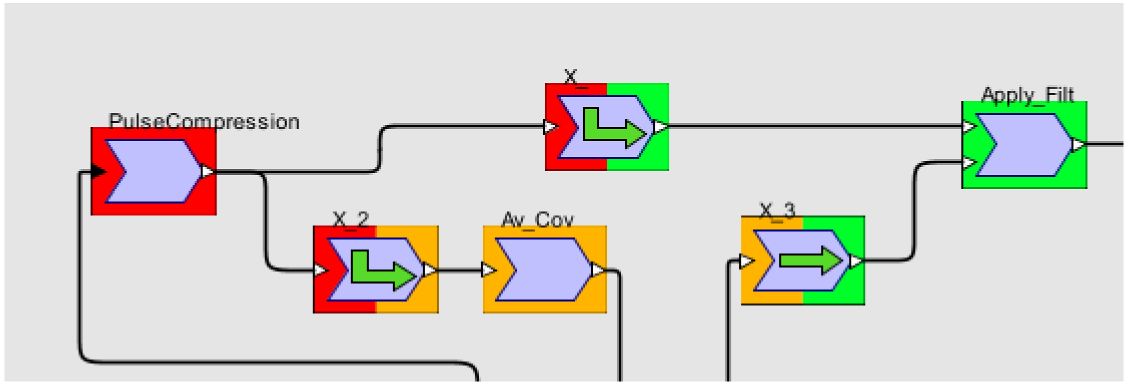




Communications

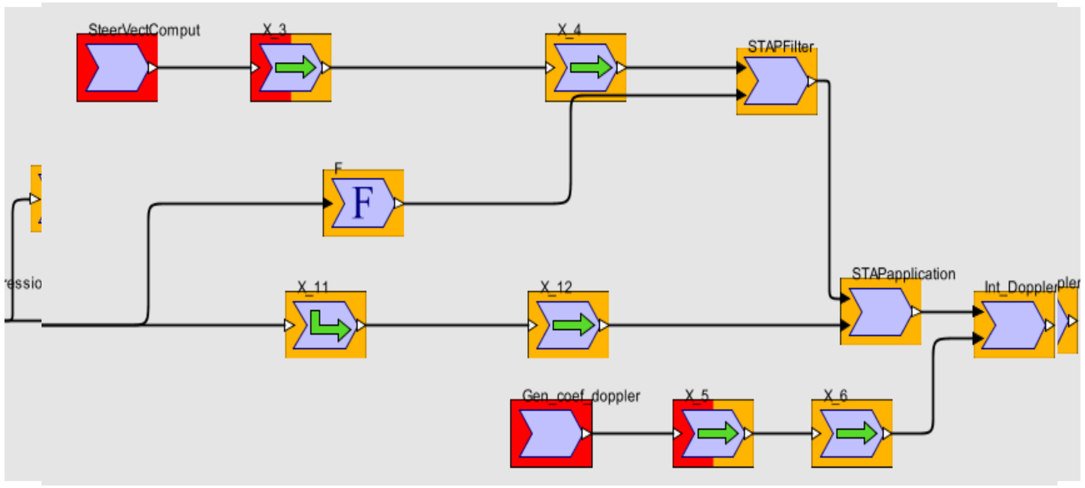
Legal data transfers

Data reorganization



- Memory estimation

Fusion



Static scheduling enabling code generation

MINES ParisTech

THALES

Code generation

Various code generators

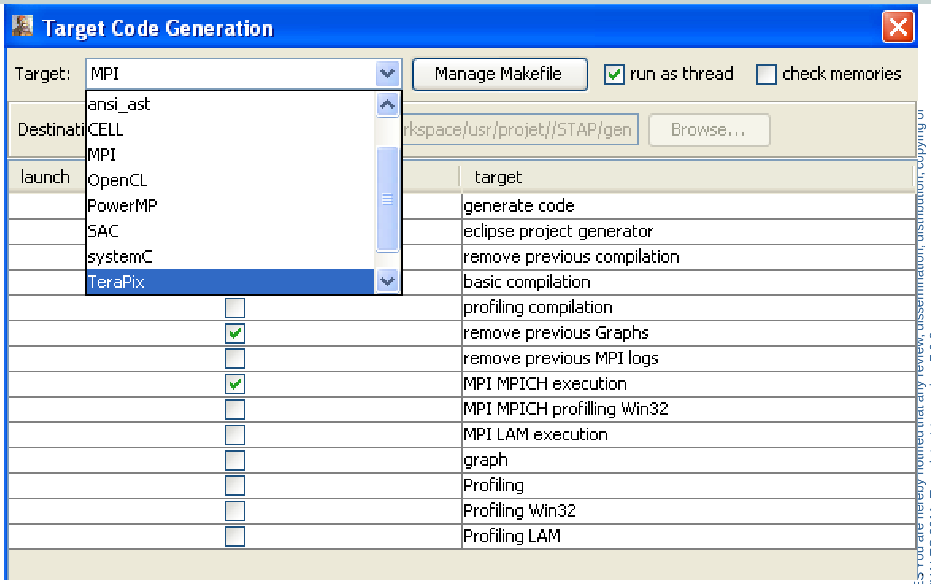
- Sequential C for functional debug
- MPI
- SystemC
- OpenCL/GPU
- Custom in-house FPGA architecture (Terapix, …)

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


Target Code Generation

Target: MPI — Manage Makefile — [✓] run as thread — [ ] check memories

Target list: ansi_ast, CELL, MPI, OpenCL, PowerMP, SAC, systemC, TeraPix

Destinati… rkspace/usr/projet//STAP/gen — Browse...

launch	target
	generate code
	eclipse project generator
	remove previous compilation
	basic compilation
☐	profiling compilation
✓	remove previous Graphs
☐	remove previous MPI logs
✓	MPI MPICH execution
☐	MPI MPICH profilling Win32
☐	MPI LAM execution
☐	graph
☐	Profiling
☐	Profiling Win32
☐	Profiling LAM

```
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

MINES ParisTech

THALES