PIPS

An Interprocedural, Extensible, Source-to-Source Compiler Infrastructure for Code Transformations and Instrumentations

15th ACM SIGPLAN Symposium on Principles
and Practice of Parallel Programming
PPoPP 2010

For the most recent version of these slides, see:
http://www.pips4u.org

Last edited:
January 9, 2010

Whom is this Tutorial for?

- **This tutorial is relevant to people interested in:**
  - GPU or FPGA-based, hardware accelerators,
  - Hardware FPGA configuration,
  - Distributed code generation for scientific or MPSoC embedded computing,
  - Quickly developing a compiler for an exotic processor (Larrabee...),
  - And more generally to all people interested in experimenting with new program transformations, verifications and/or instrumentations.

- **This tutorial aims:**
  - To survey the features available in PIPS, using examples
  - To describe key internal data structures
  - To show how to implement new transformations
  - To introduce a few ongoing projects.

Once upon a Time...

- **1823:** J.B.J. Fourier, « Analyse des travaux de l'Académie Royale des Sciences pendant l'année 1823 »
- **1936:** Theodor Motzkin, « Beiträge zur Theorie der linearen Ungleichungen »
- **1947:** George Dantzig, Simplex Algorithm
- Linear Programming, Integer Linear Programming

$$\exists?\ Q \text{ s.t. } \{x \mid \exists\ y\ P(x,y)\} = \{x \mid Q(x)\}$$

Once upon a Time... I.0.4

- **1984:** Rémi Triolet, interprocedural parallelization, convex array regions
- **1987:** François Irigoin, tiling, control code generation
- **1988:** PIPS begins...
- **1991:** Corinne Ancourt, code generation for data communication
- **1993:** Yi-qing Yang, dependence abstractions
- **1994:** Lei Zhou, execution cost models
- **1996:** Arnauld Leservot, Presburger arithmetic
- **1996:** Fabien Coelho, HPF compiler, distributed code generation
- **1996:** Béatrice Creusillet, must/exact regions, in and out regions, array privatization, coarse grain parallelization
- **1999:** Julien Zory, expression optimization

Ten years ago...
Why do we need this today?
→ Heterogeneous computing!

~~In the West~~ In France... *I.0.5*

- **2002:** Nga Nguyen, array bound check, alias analysis, variable initialization
- **2002:** Youcef Bouchebaba, tiling, fusion and array reallocation
- **2003:** C parser, MMX vectorizer, VHDL code generation
- **2004:** STEP Project: OpenMP to MPI translation
- **2005:** Ter@ops Project: XML code modelization, interactive compilation
- **2006:** Comap Project, code generation for programmable hardware accelerator
- **2007:** HPC Project startup is born
- **2008:** FREIA Project: heterogeneous computing, FPGA-based hardware accelerators
- **2009:** Par4All initiative + Ronan Keryell: CUDA code generation
- **2010:** OpenGPU Project: CUDA and OpenCL code generation

  SCALOPES: code generation for embedded parallel architectures

What is PIPS?

- **Source-to-source Fortran and C compiler, written in C**
  - Maintained by MINES ParisTech, TELECOM Bretagne / SudParis and HPC Project
- **Includes free Flex/Bison-based parsers for C and Fortran**
- **Internal representation with powerful iterators (30K lines)**
- **Compiler passes (300K+ lines and growing)**
  - Static interprocedural analyses
  - Code transformations
  - Instrumentations (dynamic analyses)
  - Source code generation
- **Main drivers of the PIPS effort:**
  - Automatic interprocedural parallelization
  - Heterogeneous computing
  - Code safety

Teams Currently Involved in PIPS

I.0.7

- **MINES ParisTech (Fontainebleau, France)**
  - Corinne Ancourt, Fabien Coelho, Laurent Daverio, **François Irigoin**
  - Mehdi Amini, Amira Mensi
- **TELECOM Bretagne (Brest, France)**
  - Stéphanie Even
  - **Serge Guelton**
- **TELECOM SudParis (Evry, France)**
  - Alain Muller, **Frédérique Silber-Chaussumier**
- **HPC Project (Paris, France)**
  - Béatrice Creusillet, Johan Gall, Ronan Keryell, Raphaël Roosz, Pierre Villalon
  - Mehdi Amini

Past contributors: CEA, ENS Cachan,...

Why PIPS? (1/2)

I.0.8

- **A source-to-source interprocedural translator, because:**
  - Parallelization techniques tend to be source transformations
  - Outputs of all optimization and compilation steps, can be expressed in C
  - Allows comparison of original and transformed codes, easy tracing and IR debugging
  - Instrumentation is easy, as well as transformation combinations.
- **Some alternatives:**
  - Polaris, SUIF: not maintained any longer
  - GCC has no source-to-source capability; entrance cost; low-level SSA internal representation.
  - Open64's 5 IRs are more complex than we needed
  - CETUS (Purdue), OSCAR (Waseda), PoCC (INRIA), Rose (LLNL)...
  - LLVM (Urbana-Champaign)

Why PIPS? (2/2)

- **A new compiler written in a modern language?**
  - No memory leaks
  - Standard library
  - Easy embedding and extension
- **Or a time-proven, feature-rich, existing Fortran and C framework?**
  - Inherit lots of static and dynamic analyses, transformations, code generations
  - Designed as a framework, easy to extend
  - Static and dynamic typing to offer powerful iterators
  - Global interprocedural consistence between analyses and transformations
  - Persistence and Python binding for more extensibility
  - Script and window-based user interfaces
  - And there are tools for memory leaks: e.g. valgrind

→ **Best alternative is to reuse existing time-proven software!**

Download and License

- **PIPS is free software**
  - Distributed under the terms of the GNU Public License (GPL) v3+.
- **It is available primarily in source form**
  - http://pips4u.org/getting-pips
  - PIPS has been compiled and run under several kinds of Unix-like (Solaris, Linux).
  - Currently, the preferred environment is **amd64 GNU/Linux.**
  - To facilitate installation, a **setup script** is provided to automatically check and/or fetch required dependencies (eg. the Linear and Newgen libraries)

    → *More info on Slide III.1.2*
  - Support is available via irc, e-mail and a Trac site.
- **Unofficial Debian GNU/Linux packages**
  - Source and binary packages for Debian Sid (unstable) on x86 and amd64: http://ridee.enstb.org/debian/info.html

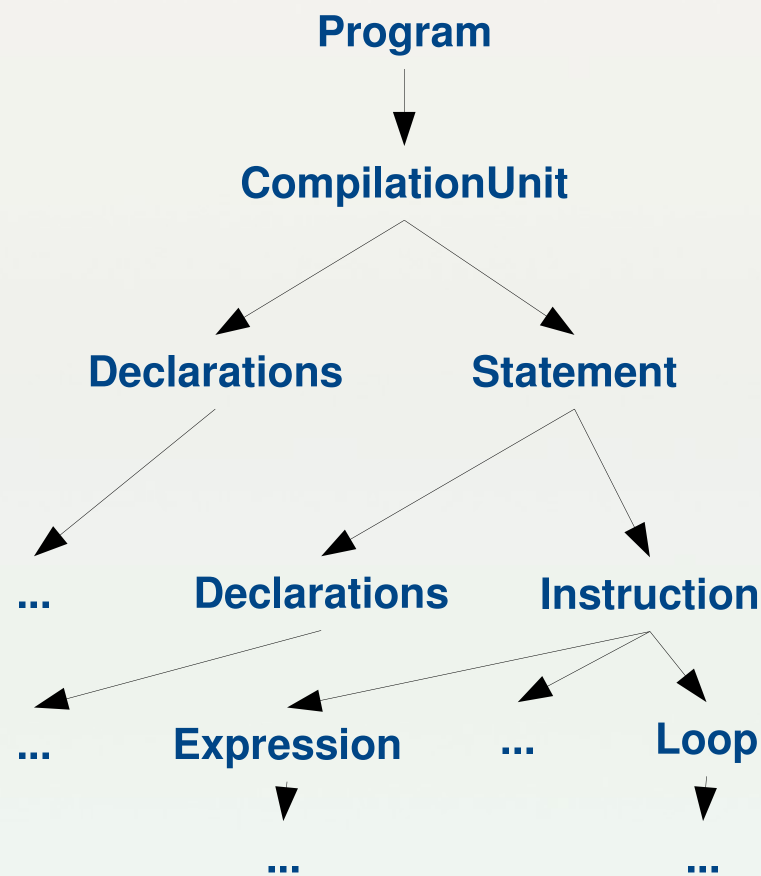
A First Example: Source-to-Source Compilation


```
int
main (void)
{
  int i,j,c,a[100];

  c = 2;
  /* a simple parallel loop */
  for (i = 0;i<100;i++)
    {
      a[i] = c*a[i]+(a[i]-1);
    }
}
```

```
delete intro_example01
create intro_example01 \
    intro_example01.c

apply UNSPLIT

close
quit
```

```
int main(void)
{
   int i, j, c, a[100];

   c = 2;
   /* a simple parallel loop */
   for(i = 0; i <= 99; i += 1)
      a[i] = c*a[i]+a[i]-1;
}
```

Explicit destruction of workspace



Simple Tree-Based IR
As closely associated with original program structure as possible for regeneration of source code

Source-to-Source Parallelization

I.0.12

```
int foo(void)
{
  int i;
  double t, s=0., a[100];
  for (i=0; i<50; ++i) {
    t = a[i];
    a[i+50] = t + (a[i]+a[i+50])/2.0;
    s = s + 2 * a[i];
  }
  return s;
}
```

```
delete intro_example02
create intro_example02 intro_example02.c

setproperty PRETTYPRINT_SEQUENTIAL_STYLE "do"

apply PRIVATIZE_MODULE[foo]
apply INTERNALIZE_PARALLEL_CODE
apply OMPIFY_CODE[foo]
display PRINTED_FILE[foo]

quit
```

```
int foo(void)
{
   int i;
   double t, s = 0., a[100];
#pragma omp parallel for private(t)
   for(i = 0; i <= 49; i += 1) {
      t = a[i];
      a[i+50] = t+(a[i]+a[i+50])/2.0;
   }
#pragma omp parallel for reduction(+:s)
   for(i = 0; i <= 49; i += 1)
      s = s+2*a[i];
   return s;
}
```

Oops, low level.
Encapsulation needed!

Q: Garbage Out? A: Garbage In!

I.0.13

```
int foo(void)
{
  int i;
  double t, s, a[100];
  for (i=0; i<50; ++i) {
    t = a[i];
    a[i+50] = t + (a[i]+a[i+50])/2.0;
    s = s + 2 * a[i];
  }
  return 0;
}
```

```
int foo(void)
{
   int i;
   double t, s, a[100];
#pragma omp parallel for private(t)
   for(i = 0; i <= 49; i += 1) {
      t = a[i];
      a[i+50] = t+(a[i]+a[i+50])/2.0;
   }
#pragma omp parallel for private(s)
   for(i = 0; i <= 49; i += 1)
      s = s+2*a[i];
   return 0;
}
```

private(s)?

```
int foo(void)
{
   return 0;
}
```

Example: Array Bound Checking


```
real function sum(n, a)
real s, a(100)
s  = 0.
do i = 1, n
   s = s + 2. * a(i)
enddo
sum = s
end
```

```
delete intro_example03
create intro_example03 intro_example03.f

setproperty PRETTYPRINT_STATEMENT_NUMBER FALSE
activate MUST_REGIONS

apply ARRAY_BOUND_CHECK_TOP_DOWN
apply UNSPLIT

close
quit
```

or: ARRAY_BOUND_CHECK_BOTTOM_UP

```
!!
!! file for intro_example03.f
!!
      REAL FUNCTION SUM(N, A)
      REAL S, A(100)
      IF (101.LE.N) STOP 'Bound violation:, READING,  array SUM:A, upper
     & bound, 1st dimension'
      S = 0.
      DO I = 1, N
         S = S+2.*A(I)
      ENDDO
      SUM = S
      END
```

Test hoisted out of the loop

User Interfaces

- **Scripting:**
  - tpips: standard interface, used in previous examples
- **Shell command:**
  - pipscc
  - Pips, Init, Display, Delete,...
- **GUI:**
  - gpips: under development
  - wpips, epips, jpips: not useful for real work
- **Programming + Scripting:**
  - pyps (+ iPython): future interface?

Scripting PIPS: tpips

- **Tpips can be interactive or scripted**
- **With tpips, you can:**
  - Manage workspaces
    - create, delete, open, close
  - Set properties
  - Activate rules
  - Apply transformations
  - Display resources
  - Execute shell commands
  - …

```
delete intro_example03
create intro_example03 intro_example03.f

setproperty PRETTYPRINT_STATEMENT_NUMBER FALSE
activate MUST_REGIONS

apply ARRAY_BOUND_CHECK_TOP_DOWN
apply UNSPLIT

close
quit
```

- **All internal pieces of information can be displayed**
- **Tpips User Manual:**
  - See http://pips4u.org/doc/manuals (HTML or PDF)

A Python Binding for PIPS: pyps

I.0.17

- **A new tool for scripting PIPS:**
  - More flexible than tpips
  - Requires knowledge of the **Python** language
  - Access PIPS objects and transformations using a popular and simple programming language
  - Ongoing work…

```
delete intro_example03
create intro_example03 intro_example03.f

setproperty PRETTYPRINT_STATEMENT_NUMBER FALSE
activate MUST_REGIONS

apply ARRAY_BOUND_CHECK_TOP_DOWN
apply UNSPLIT

close
quit
```

```
from pyps import *

# create the pips workspace
w = workspace(["intro_example03.f"])

w.set_property(PRETTYPRINT_STATEMENT_NUMBER=False)
w.activate('MUST_REGIONS')

w.all.array_bound_check_top_down()

w.all.unsplit()
```

Outline

I. What is PIPS ?

II. Using PIPS

1. Static analyses
2. Transformations (loop, ...)
3. Instrumentations (including dynamic analyses)

III. Extending PIPS

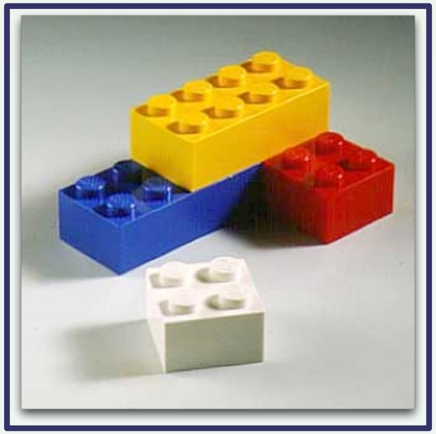
1. PIPS architecture
2. Internal representation and iterators
3. PIPS development environment

IV. A Python PIPS API

V. Ongoing projects based on PIPS

- STEP (OpenMP → MPI), CUDA generation, SSE,...

VI. Conclusion



II. Using PIPS

II. Using PIPS

Using PIPS

- **Interprocedural static analyses**
  - Semantic
  - Memory effects
  - Dependences
  - Array Regions
- **Transformations**
  - Loop transformations
  - Code transformations
    - Restructuration
    - Cleaning
  - Memory re-allocations
- **Instrumentation/ dynamic analyses**
  - Array bound checking
  - Alias checking
  - Variable initialization
- **Source code generation**
  - OpenMP
  - MPI

- Property verification: buffer overflow,...
- Optimization
- Parallelization
- Maintenance
- Reuse
- Debugging
- Conformance to standards
- Heterogeneous computing: GPU/CUDA
- Visual programming
- Interactive compilation

A variety of goals: well beyond parallelization!

- **Code modelling**
- **Prettyprint**
  - Source code [with analysis results]
  - Call tree, call graph
  - Interprocedural control flow graph

Key Concepts by Example

II.0.3

```
void bar(int n, double a[n], b[n], int i)
```

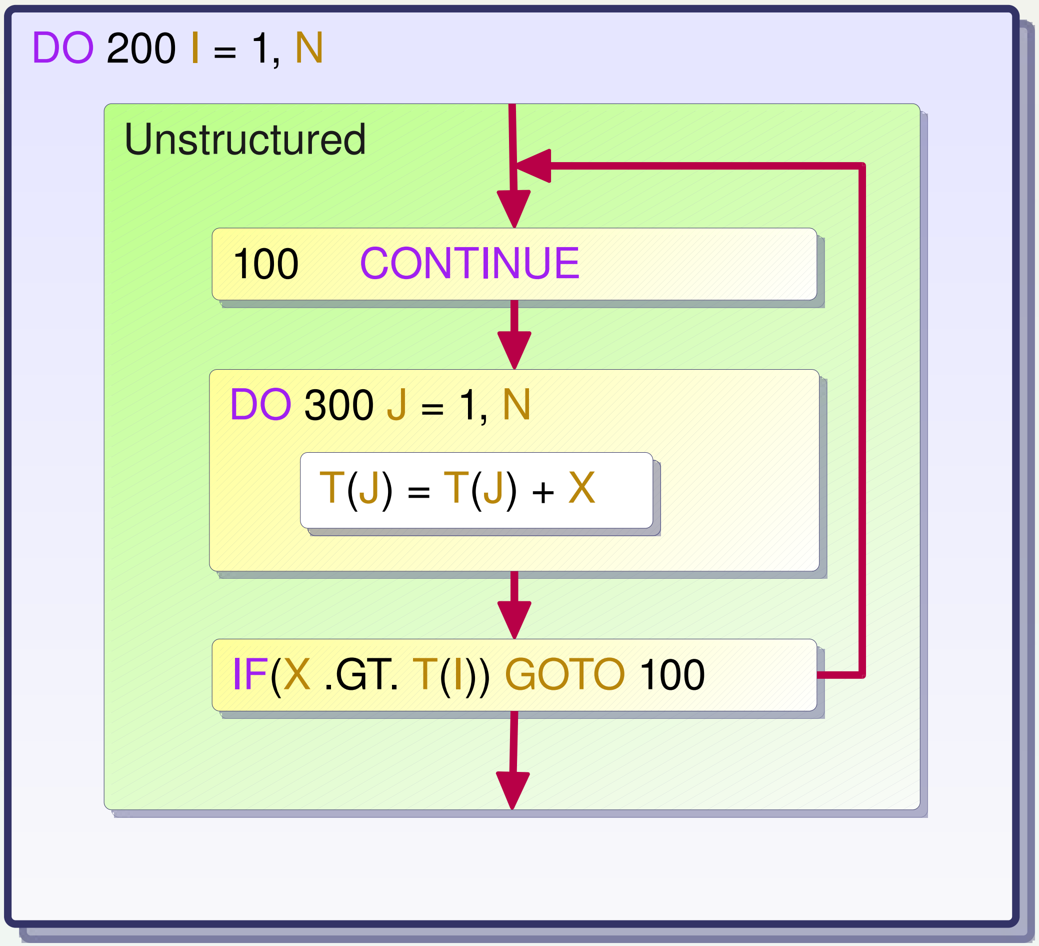
```
void foo(int n, double a[n],
double b[n])
{
  int j = 1;
  // precondition: j=1
  if(j>n) {
    // precondition: j=1 ^ j>n
    for(i=1; i<n-1; i++)
      // precondition: j=1 ^
j>n ^ 0<=i<n
      bar(n, a, b, i);

  }
}
```

Proper Read	Cumul. Read	Proper Written	Cumul. Written	bar body
b[i]	b[*]	a[i]	a[*]	{a[i]=b[i]*b[i];}
a[i], b[i]	a[*], b[*]	a[i]	a[*]	{a[i]=a[i]+b[i];}
a[i-1], b[i]	a[*], b[*]	a[i]	a[*]	{a[i]=a[i-1]+b[i];}
b[i-1:i+1]	b[*]	a[i]	a[*]	{a[i] = b[i-1]+b[i]+b[i+1];}
a[i], b[0:i]	a[*], b[*]	a[i]	a[*]	{ int k; a[i]=0; for(k=0; k<= i; k++) a[i] += b[k]; }
∅	∅	a[i-1:i+1]	a[*]	{a[i-1]=-1.; a[i] = 0.; a[i+1] = 1;}

Key Concepts by Example (cont.)


```
      DO 200 I = 1, N
100     CONTINUE
        DO 300 J = 1, N
          T(J) = T(J) + X
300     CONTINUE
        IF(X .GT. T(I)) GOTO 100
200   CONTINUE
```

Hierarchical Control Flow Graph (HCFG)



HCFG enables
structural induction over AST:

$\mathcal{F}(s1;s2) = \mathcal{C}(\mathcal{F}(s1), \mathcal{F}(s2))$

Static Analyses

Semantics:

- Transformers
  - Predicate about state transitions
- Preconditions
  - Predicate about state

Memory Effects:

- Read/Write effects
- In/Out effects
- Read/Write convex array regions
- In convex array regions
- Out convex array regions

Dependences:

- Use/def chains
- Region-based use/def chains
- Dependences (levels, cones)

Experimental Analyses:

- Flow-sensitive, context-insensitive pointer analysis
- Complexity
- Total preconditions

Principle: Each Function is Analyzed Once
Summaries must be built

Preconditions

II.1.2

- Affine predicates on scalar variables
  - Integer, float, complex, boolean, string
- Options:
  - Trust array references or Transformer in context, …
- Innovative fix point operator

Includes symbolic ranges

```
// P() {}
int main()
{
   float a[10][10], b[10][10], h;
   int i, j;
// P() {}
   for(i = 1; i <= 10; i += 1)
// P(i,j) {1<=i, i<=10}
      for(j = 1; j <= 10; j += 1)
// P(i,j) {1<=i, i<=10, 1<=j, j<=10}
         b[i][j] = 1.0;
// P(i,j) {i==11, j==11}
   h = 2.0;
// P(h,i,j) {2.0==h, i==11, j==11}
   func1(10, 10, a, b, h);
// P(h,i,j) {2.0==h, i==11, j==11}
   for(i = 1; i <= 10; i += 1)
// P(h,i,j) {2.0==h, 1<=i, i<=10}
      for(j = 1; j <= 10; j += 1)
// P(h,i,j) {2.0==h, 1<=i, i<=10, 1<=j, j<=10}
         fprintf(stderr, "a[%d] = %f \n", i, a[i][j]);
}
```

Preconditions (cont.)

- Interprocedural analysis:
  - Summary transformer, summary precondition
  - Top-down analysis

Summary Precondition

```
//  P() {2.0==h, m==10, n==10}
void func1(int n, int m, float a[n][m], float b[n][m], float h)
{
   float x;
   int i, j;
//  P() {2.0==h, m==10, n==10}
   for(i = 1; i <= 10; i += 1)
//  P(i,j,x) {2.0==h, m==10, n==10, 1<=i, i<=10}
      for(j = 1; j <= 10; j += 1) {
//  P(i,j,x) {2.0==h, m==10, n==10, 1<=i, i<=10, 1<=j, j<=10}
         x = i*h+j;
//  P(i,j,x) {2.0==h, m==10, n==10, 1<=i, i<=10, 1<=j, j<=10}
         a[i][j] = b[i][j]*x;
      }
}
```

```
//  P() {}
int main()
{
   float a[10][10], b[10][10], h;
   int i, j;
//  P() {}
   for(i = 1; i <= 10; i += 1)
//  P(i,j) {1<=i, i<=10}
      for(j = 1; j <= 10; j += 1)
//  P(i,j) {1<=i, i<=10, 1<=j, j<=10}
         b[i][j] = 1.0;
//  P(i,j) {i==11, j==11}
   h = 2.0;
//  P(h,i,j) {2.0==h, i==11, j==11}
   func1(10, 10, a, b, h);
//  P(h,i,j) {2.0==h, i==11, j==11}
   for(i = 1; i <= 10; i += 1)
//  P(h,i,j) {2.0==h, 1<=i, i<=10}
      for(j = 1; j <= 10; j += 1)
//  P(h,i,j) {2.0==h, 1<=i, i<=10, 1<=j, j<=10}
         fprintf(stderr, "a[%d] = %f \n", i, a[i][j]);
}
```

Call site

Affine Transformers, Preconditions and Summarization

- **Abstract store: precondition $P(\sigma_0, \sigma)$ or range($P(\sigma_0, \sigma)$)**
- **Abstract command: transformer $T(\sigma, \sigma')$**

```
foo()
{
  // P
  bar(n); // T = translate_foo (T_bar )
  // P' = P o T
}
```

```
// R
 bar(m-1); // T = translate_X(T_bar )
```

```
// Q
 bar(i+j); // T = translate_Y(T_bar )
```

```
// T_bar  = T_1 o T_2
void bar(int i)
{
  // P_1=union(translate_foo (P), translate_Y(Q), translate_X(R))
  S1; // T_1
  // P_2 = P_1 o T_1 (i.e. P_2 = T_1(P_1))
  S2; // T_2
}
```

Memory Effects

- **Used and def variables**
  - **Read** or **Written**
  - **May** or **Exact**
  - **Proper**, **Cumulated** or **Summary**

Proper

```
//      <must be read   >: n
//      <must be written>: i
  for(i = 1; i <= n; i += 1) {

//      <must be read   >: m n
//      <must be written>: j
    for(j = 1; j <= m; j += 1) {

//      <must be read   >: h i j m n
//      <must be written>: x
      x = i*h+j;

//      <must be read   >: b[i][j] i j m n x
//      <must be written>: a[i][j]
      a[i][j] = b[i][j]*x;
}}
```

Summary

```
//          <may be read    >: b[*][*] h
//          <may be written >: a[*][*]
//          <must be read   >: m n
func1(int n, int m, float a[n][m], float b[n][m], float h)
{
  float x;
  int i,j;
//          <may be read    >: b[*][*] h i j m x
//          <may be written >: a[*][*] j x
//          <must be read   >: n
//          <must be written>: i
  for(i = 1; i <= n; i += 1)
    for(j = 1; j <= m; j += 1) {
      x = i*h+j;
      a[i][j] = b[i][j]*x;
  }}
```

Cumulated

Convex Array Regions

- Bottom-up refinement of effects for array elements
- Polyhedral approximation of referenced array elements

```
//  <a[PHI1][PHI2]-W-EXACT-{1<=PHI1, PHI1<=n, 1<=PHI2, PHI2<=m, m==10, n==10}>
//  <b[PHI1][PHI2]-R-EXACT-{1<=PHI1, PHI1<=n, 1<=PHI2, PHI2<=m, m==10, n==10}>
void func1(int n, int m, float a[n][m], float b[n][m], float h)
{
  float x;
  int i,j;

//  <a[PHI1][PHI2]-W-EXACT-{1<=PHI1, PHI1<=n, 1<=PHI2, PHI2<=m, m==10, n==10}>
//  <b[PHI1][PHI2]-R-EXACT-{1<=PHI1, PHI1<=n, 1<=PHI2, PHI2<=m, m==10, n==10}>
  for(i = 1; i <= n; i += 1)

//  <a[PHI1][PHI2]-W-EXACT-{PHI1==i, 1<=PHI2, PHI2<=m, m==10, n==10, 1<=i, i<=n}>
//  <b[PHI1][PHI2]-R-EXACT-{PHI1==i, 1<=PHI2, PHI2<=m, m==10, n==10, 1<=i, i<=n}>
    for(j = 1; j <= m; j += 1) {
      x = i*h+j;

//  <a[PHI1][PHI2]-W-EXACT-{PHI1==i, PHI2==j, m==10, n==10, 1<=i, i<=10, 1<=j, j<=10}>
//  <b[PHI1][PHI2]-R-EXACT-{PHI1==i, PHI2==j,m==10, n==10, 1<=i, i<=10, 1<=j, j<=10}>
      a[i][j] = b[i][j]*x;
    }}
```

Interprocedural preconditions are used

A triangular iteration space could be used as well

Convex Array Regions: Use Transformers and Preconditions

- **Regions: Functions from stores σ to sets of elements φ for arrays A, …**
- **Functions $\varphi = r_A(\sigma)$ or function graphs $R_A(\varphi, \sigma)$**
- **Approximation: MAY, MUST, EXACT**
- **Use transformers $T(\sigma, \sigma')$ and preconditions $P(\sigma) = range(P(\sigma_0, \sigma))$**
  - Note: σ_0 is the function initial state

Assume S does not use nor define elements of A

```
// P(σ)

// rA(σ) : σ → { φ | RA(φ, σ) }

S: i++; // T(σ, σ')

// rA(σ') : σ' → { φ | R'A(φ, σ') }

S': a[i] = ...; // T(σ', σ'')
```

Exact quantifier elimination?

Quantifier elimination

$$R_A(\varphi, \sigma) = \{ (\varphi, \sigma) \mid \exists \sigma' \ \ T(\sigma, \sigma') \wedge R'_A(\varphi, \sigma') \wedge P(\sigma) \}$$

IN and OUT Convex Array Regions

- **IN convex array region for Statement S**
  - Memory locations whose values are used by S before they are defined
- **OUT convex array region for S**
  - Memory locations defined by S, and whose values are used later by the program
  - Sometimes surprising… when no explicit continuation exists: garbage in, garbage out

Non convex regions?

```
//  <b[PHI1][PHI2]-IN-EXACT-{ 1<=PHI1, PHI1<=n,  1<=PHI2, PHI2<=m,
//        m==10, n==10}>

//  <a[PHI1][PHI2]-OUT-EXACT-{ 1<=PHI1, PHI1<=10, 1<=PHI2, PHI2<=10,
//        m==10, n==10}>

S: for(i = 1; i <= n; i += 1)
      for(j = 1; j <= m; j += 1) {
         x = i*h+j;
        a[i][j] = b[i][j]*x;
      }
```

Requires non-monotonic operators: MUST or EXACT regions
IN(S1;S2) = IN(S1) U (READ(S2) – WRITE(S1))

Data Dependence

- **Several dependence test algorithms:**
  - Fourier-Motzkin with different information:
    - rice_fast_dependence_graph
    - rice_full_dependence_graph
    - rice_semantics_dependence_graph
  - Properties
    - Read-read dependence arcs
- **Dependence abstractions:**
  - Dependence level
  - Dependence cone
    - Includes uniform dependencies
- **Prettyprint dependence graph:**
  - Use-def chains
  - Dependence graph

My parallel loop is still sequential:

Why?

Dependence test?

Look at the dependence graph?

My parallel loop is still sequential:

Why?

Array Privatization?

Complexity

- **Symbolic approximation of execution cost: polynomials**

```
//                                        1721 (SUMMARY)
void func1(int n, int m, float a[n][m], float b[n][m], float h)
```

```
//                                17*m.n + 3*n + 2 (SUMMARY)
void func1(int n, int m, float a[n][m], float b[n][m], float h)
{
   float x;
   int i, j;
//                                17*m.n + 3*n + 2 (DO)
   for(i = 1; i <= n; i += 1)
//                                17*m + 3 (DO)
      for(j = 1; j <= m; j += 1) {
//                                 6 (STMT)
         x = i*h+j;
//                                10 (STMT)
         a[i][j] = b[i][j]*x;
      }
}
```

Application: complexity comparison before and after constant propagation.

P() {m==10, n==10}

Based on a parametric cost table

Loop Transformations

- **Loop Distribution**
- **Index set splitting**
- **Loop Interchange**
- **Hyperplane method**
- **Loop Normalization**
- **Strip Mining**
- **Tiling**
- **Full/Partial Unrolling**
- **Parallelizations**

```
apply PARTIAL_EVAL[convol]
apply LOOP_TILING[convol]
apply FULL_UNROLL[convol]
apply PARTIAL_EVAL[convol]
apply SCALARIZATION[convol]
display PRINTED_FILE[convol]
```

- **Tiling example with convol**

```
void convol(int isi, int isj, float new_image[isi][isj], float image[isi][isj],
int ksi, int ksj, float kernel[ksi][ksj])
{
   int i, j, ki, kj;
   int i_t, j_t; float __scalar__0;          //PIPS generated variables
l400:
   for(i_t = 0; i_t <= 3; i_t += 1)
      for(j_t = 0; j_t <= 3; j_t += 1)
         for(i = 1+128*i_t; i <= MIN(510, 128+128*i_t); i += 1)
            for(j = 1+128*j_t; j <= MIN(128+128*j_t, 510); j += 1) {
               __scalar__0 = 0.;
l200:          __scalar__0 = __scalar__0+image[i-1][j-1]*kernel[0][0];
               __scalar__0 = __scalar__0+image[i-1][j]*kernel[0][1];
               __scalar__0 = __scalar__0+image[i-1][j+1]*kernel[0][2];
               __scalar__0 = __scalar__0+image[i][j-1]*kernel[1][0];
               __scalar__0 = __scalar__0+image[i][j]*kernel[1][1];
               __scalar__0 = __scalar__0+image[i][j+1]*kernel[1][2];
               __scalar__0 = __scalar__0+image[i+1][j]*kernel[2][1];
               __scalar__0 = __scalar__0+image[i+1][j+1]*kernel[2][2];
               __scalar__0 = __scalar__0/9;
               new_image[i][j] = __scalar__0; }}
```

Loop Parallelization

- **Allen & Kennedy**
- **Coarse grain**
- ***Nest parallelization***

```
      PROGRAM NS
      PARAMETER (NVAR=3,NXM=2000,NYM=2000)
      REAL PHI(NVAR,NXM,NYM),PHI1(NVAR,NXM,NYM)
      REAL PHIDES(NVAR,NYM)
      REAL DIST(NXM,NYM),XNOR(2,NXM,NYM),SGN(NXM,NYM)
      REAL XCOEF(NXM,NYM),XPT(NXM),YPT(NXM)


!$OMP  PARALLEL DO PRIVATE(I,PX,PY,XCO)
     DO J = 2, NY-1

!$OMP  PARALLEL DO PRIVATE(PX,PY,XCO)
        DO I = 2, NX-1
           XCO = XCOEF(I,J)
           PX = (PHI1(3,I+1,J)-PHI1(3,I-1,J))*H1P2
           PY = (PHI1(3,I,J+1)-PHI1(3,I,J-1))*H2P2
           PHI1(1,I,J) = PHI1(1,I,J)-DT*PX*XCO
           PHI1(2,I,J) = PHI1(2,I,J)-DT*PY*XCO
        ENDDO
     ENDDO
     END
```

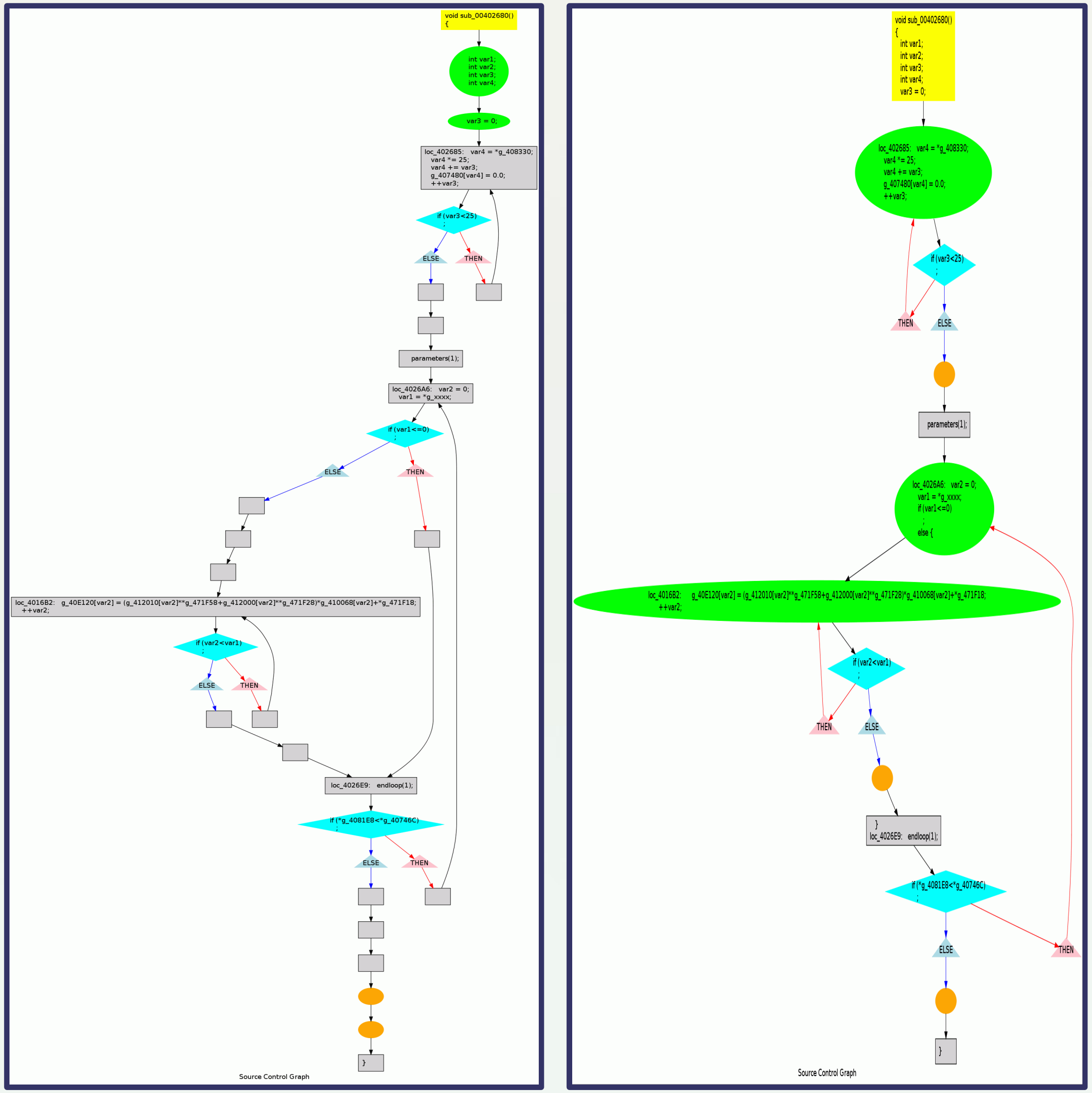
Code Transformation Phases (1)

- **Three-address code**
  - *Atomizers*
  - Two-address code
- **Reduction recognition**
- **Expression optimizations:**
  - Common subexpression elimination
  - Forward substitution
  - Invariant code motion
  - Induction variable substitution
- **Restructuring**
  - Code flattening
  - Restructure control
  - Split initializations
- **Memory optimizations:**
  - Scalar privatization
  - Array privatization from regions
  - Array/Scalar expansion
  - *Scalarization*

Code Transformation Phases (2)

- **Cloning**
- **Inlining**
- **Outlining**
- **Partial evaluation from preconditions**
  - Constant propagation + evaluation
- **Dead code elimination**
  - Redundant test elimination
  - Use-def elimination
- **Control restructuration**
  - Hierarchization
  - if/then/else restructuring
  - Loop recovery
  - For- to do-loop

A hierarchization example :



Inlining and Outlining

II.2.5

```
void convol(int n, int a[n][n], int b[n][n], int
kernel[3][3])
{
   int i, j;
   for(i = 0; i <= n-1; i += 1)
      for(j = 0; j <= n-1; j += 1) {
         {
            int k, l;
            b[i][j] = 0;
            for(k = 0; k <= 2; k += 1)
               for(l = 0; l <= 2; l += 1)
                  b[i][j] += a[i+k-1][j+l-1]*kernel[k][l];
         }
      }
}
```

```
void convol_outlined(int i, int n, int a[n][n], int
b[n][n], int kernel[3][3])
{
   //PIPS generated variable
   int j;
l99996:
   for(j = 0; j <= n-1; j += 1) {
      {
         int k, l;
         b[i][j] = 0;
l99997:
         for(k = 0; k <= 2; k += 1)
l99998:
            for(l = 0; l <= 2; l += 1)
               b[i][j] += a[i+k-1][j+l-1]*kernel[k][l];
      }
   }
}
```

```
void convol(int n, int a[n][
int kernel[3][3])
{
   int i, j;
l99995:
   for(i = 0; i <= n-1; i += 1)
l99996:      convol_outlined(i, n,
kernel);
}
```

apply UNFOLDING[convol]
apply FLAG_LOOPS[convol]
setproperty OUTLINE_LABEL "l99996"
setproperty OUTLINE_MODULE_NAME "convol_outlined"
apply OUTLINE[convol]

Cloning (+ Constant Propagation + Dead Code Elimination)

II.2.6

```
# 1
int clone01(int n, int s)
{
  int r = n;
  if(s<0)
    r = n-1;
  else if(s>0)
    r = n+1;
  return r;
}
```

Imprecise summary transformer

```
int clone01_2(int n, int s)
```

```
int clone01_1(int n, int s)
```

```
int clone01_0(int n, int s)
{
  {
    {
    }
    return 0;
  }
}
```

```
// P() {}
int main()
{
  int i = 1;
// P() {i==1}
  i = clone01(i, -1);
// P(i) {0<=i, i<=2}
  i = clone01(i, 1);
// P(i) {0<=i+1, i<=3}
  i = clone01(i, 0);
}
```

Imprecise preconditions

Exact preconditions

```
// P() {}
int main()
{
  int i = 1;
// P() {i==1}
  i = clone01_0(i, -1);
// P(i) {i==0}
  i = clone01_1(i, 1);
// P(i) {i==1}
  i = clone01_2(i, 0);
}
```

Dead Code Elimination (1)

- **Suppress dead code:**
  - Redundant test elimination
  - Use preconditions to eliminate tests and simplify zero- and one-trip loops
- **Partial evaluation**
  - Interprocedural constant propagation
- **Use-def elimination**

```
int clone01_1(int n, int s)
{
  // PIPS: s is assumed a constant reaching value
  if (s!=1)
    exit(0);
  {
    int r = n;
    if (s<0)
      r = n-1;
    else if (s>0)
      r = n+1;
    return r;
  }
}
```

```
int clone01_1(int n, int s)
{
  // PIPS: s is assumed a constant reaching value
  if (1!=1)
    exit(0);
  {
    int r = 0;
    if (1<0)
      r = n-1;
    else if (1>0)
      r = 1;
    return 1;
  }
}
```

```
int clone01_1(int n, int s)
{
  // PIPS: s is assumed a constant reaching value
  ;
  {
    {
    }
    ;
    return 1;
  }
}
```

```
int clone01_1(int n, int s)
{
  {
    {
    }
    return 1;
  }
}
```

Dead Code Elimination (2)

- **Partial eval**
- **Suppress dead code**
- **Use-def elimination**

Cloning warning

```
int clone02_1(int n, int s)
{
  // PIPS: s is assumed a constant reaching value
  if (s!=1)
    exit(0);
  {
    int r = n;
    if (s<0)
      r = n-1;
    else if (s>0)
      r = n+1;
    return r;
  }
}
```

```
int clone02_1(int n, int s)
{
  // PIPS: s is assumed a constant reaching value
  if (1!=1)
    exit(0);
  {
    int r = 0;
    if (1<0)
      r = n-1;
    else if (1>0)
      r = 1;
    return 1;
  }
}
```

```
int clone02_1(int n, int s)
{
  {
    // PIPS: s is assumed a constant reaching value
    int r = 0;
    r = 1;
    return 1;
  }
}
```

```
int clone02_1(int n, int s)
{
  {
    {
      // PIPS: s is assumed a constant reaching value
      ;
    }
    ;
    return 1;
  }
}
```

Maintenance and Debugging: Dynamic Analyses

- **Uninitialized variable detection (used before set, UBS)**
- **Fortran type checking**
- **Declarations: cleaning**
- **Array resizing**
- **Fortran alias detection**
- **Array bound checking**

```
!!
!! file for scalar02.f
!!
      PROGRAM SCALAR02
      INTEGER X,Y,A,B
      EXTERNAL ir_isnan,id_isnan
      LOGICAL*4 ir_isnan,id_isnan
      STOP 'Variable SCALAR02:Y is used before set'
      STOP 'Variable SCALAR02:B is used before set'
      X = Y
      A = B
      PRINT *, X, A
      B = 1
      RETURN
      END
```

Prettyprint

II.4.1

- **Fortran 77**
  - + OpenMP directives
  - + Fortran 90 array expressions
- **Fortran 77: a long history…**
  - + HPF directives
  - + DOALL loops
  - + Fortran CRAY
  - + CMF

The results of all PIPS analyses can be prettyprinted and visualized with the source code:

activate PRINT_CODE_PRECONDITIONS
display PRINTED_FILE

- **C**
  - + OpenMP directives
- **XML**
  - Code modelling
  - Visual programming
- **Graphs**
  - Call tree, call graph
  - Use-Def chains
  - Dependence graph
  - Interprocedural control flow graph

Source Code Generation

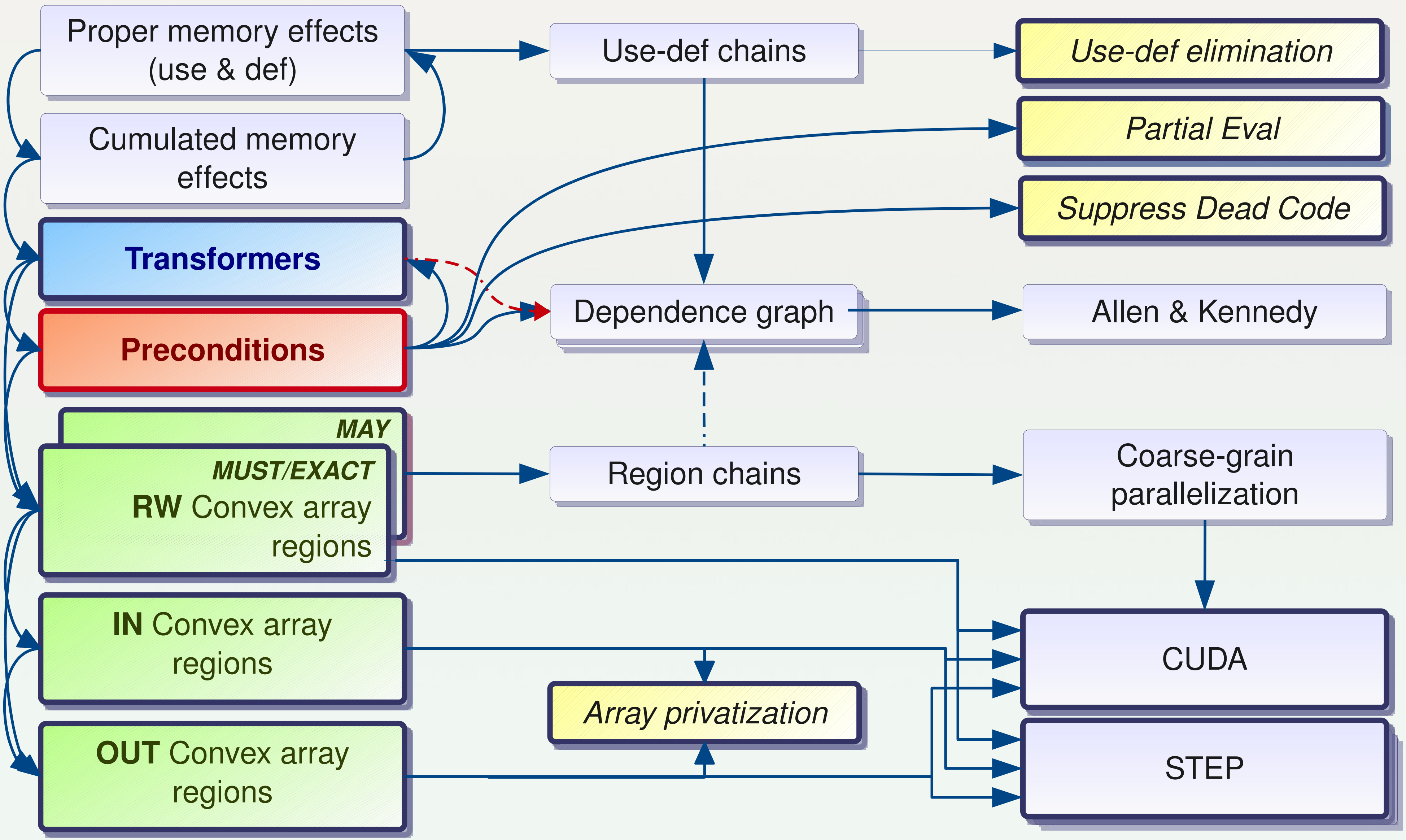
- **HPF**
  - MPI
  - PVM
- **OpenMP → MPI**
- **GPU/CUDA**
- **SSE**
- **Ongoing:**
  - OpenCL
  - FREIA

Excerpt of an image alphablending function

```
....
  SIMD_LOAD_GENERIC_V4SF(v4sf_vec1, alpha, alpha, alpha, alpha);
  SIMD_LOAD_CONSTANT_V4SF(v4sf_vec4, 1, 1, 1, 1);
  LU_IND0 = LU_IB0+MAX(INT((LU_NUB0-LU_IB0+3)/4), 0)*4;
  SIMD_SUBPS(v4sf_vec3, v4sf_vec4, v4sf_vec1);
  for(LU_IND0 = LU_IB0; LU_IND0 <= LU_NUB0-1; LU_IND0 += 4) {
    SIMD_LOAD_V4SF(v4sf_vec2, &src1[LU_IND0]);
    SIMD_MULPS(v4sf_vec0, v4sf_vec1, v4sf_vec2);
    SIMD_LOAD_V4SF(v4sf_vec8, &src2[LU_IND0]);
    SIMD_MULPS(v4sf_vec6, v4sf_vec3, v4sf_vec8);
    SIMD_ADDPS(v4sf_vec9, v4sf_vec0, v4sf_vec6);
    SIMD_SAVE_V4SF(v4sf_vec9, &result[LU_IND0]);
  }
  SIMD_SAVE_GENERIC_V4SF(v4sf_vec0, &F_03, &F_02, &F_01, &F_00);
  SIMD_SAVE_GENERIC_V4SF(v4sf_vec3, &F_13, &F_12, &F_11, &F_10);
  SIMD_SAVE_GENERIC_V4SF(v4sf_vec6, &F_23, &F_22, &F_21, &F_20);
}
```

Assembly level code

Relationships: Analyses, Transformations & Code Generation



Using PIPS: Wrap-Up

- **Analyze...**
  - to decide what parts of code to optimize
  - to detect parallelism
- **Transform...**
  - to simplify, optimize locally
  - to adjust code to memory constraints and parallel components
- **Generate code for a target architecture**
  - SSE
  - CUDA

- Interprocedural analyses
  - Preconditions, array regions, dependences, complexity
- Transformations
  - Constant propagation, loop unrolling,
  - Expression optimization, privatization, scalarization,
  - Loop parallelization, tiling, inlining, outlining,
- Prettyprints
  - OpenMP

→ What's missing? Let's add a new program transformation...

III. Extending PIPS

III. Extending PIPS

III. Extending PIPS

- **Developing a new pass (i.e. phase) within PIPS:**
  - How do I get a copy of the PIPS source code?
  - Where do I put my own source code?
  - The API: how do I manipulate the IR?
  - How do I find the existing library functions I need?
  - How do I integrate my pass in PIPS?
  - How do I add non-regression tests?
  - How do I merge my branch copy back into the trunk? (core developers only)
- **Distributed development: working with Subversion (SVN)**
- **Documentation:**
  - See the "PIPS Online Resources" slide
  - PIPS Developer Guide
  - Internal representation

Well, this is central for code reuse!

Example 1: Add a comment

III.1.1

- **Purpose: add a comment to the first statement of a module**

```
float extending01(int n, float a[n])
{
  int i;
  float s=0.;
  for(i=0;i<n;i++) {
    s += a[i]*a[i];
  }
  return s;
}
```

```
float extending01(int n, float a[n])
{
   // PPoPP 2010
   int i;
   float s = 0.;
   for(i = 0; i <= n-1; i += 1)
      s += a[i]*a[i];
   return s;
}
```

```
delete extending01
create extending01 extending01.c

setproperty PREPEND_COMMENT " PPoPP 2010"

apply PREPEND_COMMENT
display PRINTED_FILE

close
quit
```

Override default property value

Let's get PIPS!

- **Install dependencies (e.g. for Ubuntu Linux 9.10 « Karmic Koala »)**

```
$ sudo apt-get install subversion cproto flex bison indent gfortran fort77 \
libreadline5-dev tex4ht texlive-latex-extra emacs23 sun-java6-jdk python-dev
swig
```

- **Download and execute setup_pips.sh (install from SVN)**

```
$ wget http://svn.cri.ensmp.fr/svn/nlpmake/trunk/makes/setup_pips.sh
$ chmod +x setup_pips.sh
$ ./setup_pips.sh # use default options
# → ... a few minutes for downloads and compilations, with lots of warnings... at
first
$ source MYPIPS/pipsrc.sh # and add to .profile?
```

Automatic generation of header files

- **Validate build (non-regression tests)**

```
$ unset LANG # to control the behaviour of "diff"
$ cd ../validation
$ make validate
```

- **For more details, please refer to the online Installation Guide**

SVN: To Branch Or Not To Branch?

- **Developers wishing to take part to the development of PIPS can request their own branch on the PIPS SVN server**
  - Having a SVN branch allows you to make and test your developments without interfering with the trunk and, eventually, to **merge them back** into the trunk.
    - Of course, committer rights are needed!

Ask us!

- **The default PIPS setup reflects this organisation**
  - **MYPIPS/prod/pips**: working copy from the trunk
  - **MYPIPS/pips_dev**: working copy from your branch (*if applicable*)
  - **MYPIPS/validation**: test cases
- **Switching from one copy to the other**
  - It is as simple as editing environment variable **$PIPS_ROOT** in MYPIPS/pipsrc.sh.
- **Subversion >= 1.5**
  - Greatly simplifies merging changes (from the trunk and back into the trunk)

Create a New Library and its Local Header Files


```
$ cd $PIPS_ROOT/src/Libs
$ mkdir to_begin_with; ls
$ vi local.mk
$ cd to_begin_with
$ vi Makefile
$ touch to_begin_with-local.h
```

See Developer Guide, Section 10.3

```
$ ls $PIPS_ROOT/src/Libs
alias-classes     expressions      pip              semantics
array_dfg         flint            pipsdbm          static_controlize
atomizer          gpu              pipsmake         statistics
bootstrap         hpfc             preprocessor     step
callgraph         hwac             prettyprint      syntax
chains            hyperplane       prgm_mapping     text-util
complexity        icfg             properties       to_begin_with
comp_sections     instrumentation  reductions       TODO
continuation      local.mk         reindexing       top-level
control           Makefile         rice             transformations
conversion        misc             ricedg           transformer
c_syntax          movements        ri-util          wp65
effects-convex    newgen           sac
effects-generic   paf-util         safescale
effects-simple    phrase           scheduling
```

A Quick Look at the Most Important PIPS Libraries

III.1.5

pips | tpips | gpips | pyps | ...

top-level
pipsmake

Analyses
effects-***
chains
ricedg
semantics
complexity

Parsers
syntax
c_syntax
control
bootstrap
preprocessor

Transformations
transformations
expressions
hyperplane
movements
reductions
statistics

alias-classes
instrumentation

Code generation
rice
gpu
hpfc
hwac
sac
step
...

prettyprint
callgraph
icfg

ri-util (entity, module, statement, expression,...)

text-util

pipsdbm

newgen

Linear

Create a New C File: add_stuff_to_module.c

- **Step 1: mandatory source code skeleton for *pipsmake* interface**

Boolean function: phase success?

Phase name: preprend_comment

Unique argument: a module name

```
// Insert required headers here

bool prepend_comment(string mn) {

  // Use the module name mn to get the resource(s) we need
  statement s = ... ;


  // Get the value of the property containing the comment to be prepended
  string c = ... ;

  // Add comment c to the module statement s
  s = ... ;

  // Put back the updated statement module as a new resource

}
```

Fill in the New C File

- **Step 2: add headers**

```
#include "genC.h"
#include "linear.h"
#include "ri.h"
#include "ri-util.h"
#include "misc.h"
#include "pipsdbm.h"
#include "resources.h"
```

← Newgen: the bedrock
← Linear: required by ri.h
← Internal Representation Infrastructure
← Error / debug management
← Resources and persistence

```
// Insert required headers here

bool prepend_comment(string mn) {

  // Use the module name mn to get the resource(s) we need
  statement s = ... ;


  // Get the value of the property containing the comment to be prepended
  string c = ... ;

  // Add comment c to the module statement s
  s = ... ;

  // Put back the updated statement module as a new resource

}
```

Why don't we use self-contained headers or more inclusive headers?
Because programmers include too much stuff and create cycles between libraries

Fill in the New C File (cont.)

III.1.8

- **Step 3: prologue and epilogue**

```
#define PIPS_PHASE_PRELUDE(mn, debug_env_var)                        \
  (statement) db_get_memory_resource(DBR_CODE, mn, TRUE);            \
              set_current_module_statement(s);                       \
              entity mod = module_name_to_entity(mn);                \
              set_current_module_entity(mod);                        \
              debug_on(debug_env_var);                               \
              pips_debug(1, "Entering...\n");                        \
              pips_assert("Statement should be OK before...",        \
                          statement_consistent_p(s));
```

```
// Insert requ
bool prepend

  // Use this m
  statement s = PIPS_PHASE_PRELUDE(mn,
                      "PREPEND_COMMENT_DEBUG_LEVEL" );

  // Get the value of the prope
  string c = ... ;

  // Add comment c to the mo
  s = ... ;

  // Put back the new stateme

  PIPS_PHASE_POSTLUDE
```

```
#define PIPS_PHASE_POSTLUDE(s)                                  \
  pips_assert("Statement should be OK after...",               \
              statement_consistent_p(s));                       \
  pips_debug(1, "done\n"); debug_off();                         \
  DB_PUT_MEMORY_RESOURCE(DBR_CODE,                              \
                         get_current_module_name(),             \
                         s);                                    \
  reset_current_module_statement();                             \
  reset_current_module_entity();                                \
  return TRUE;
```

Fill in the New C File (cont.)

- **Step 4: find the proper IR function to add a comment**

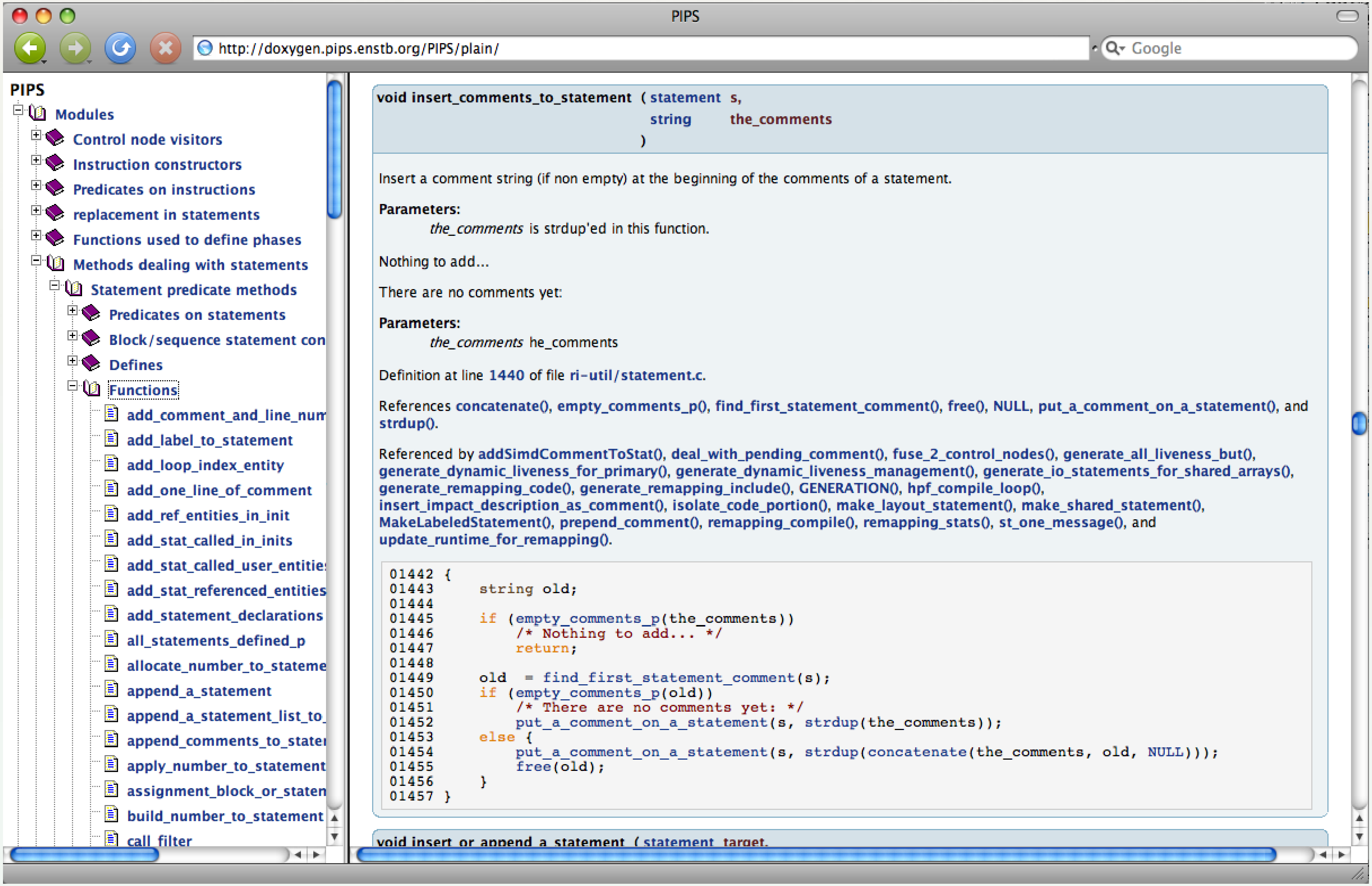
Query Doxygen ?
Query pipsdev / IRC?
Query TAGS?
Use Eclipse?
Query GoogleCode?

```
bool prepend_comment(string mn) {

  // Use this module name to get the resources we need
  statement s = PIP

  // Get the value of
  string c = ... ;

  // Add comment c
  s = ... ;

  // Put back the ne

  PIPS_
}
```

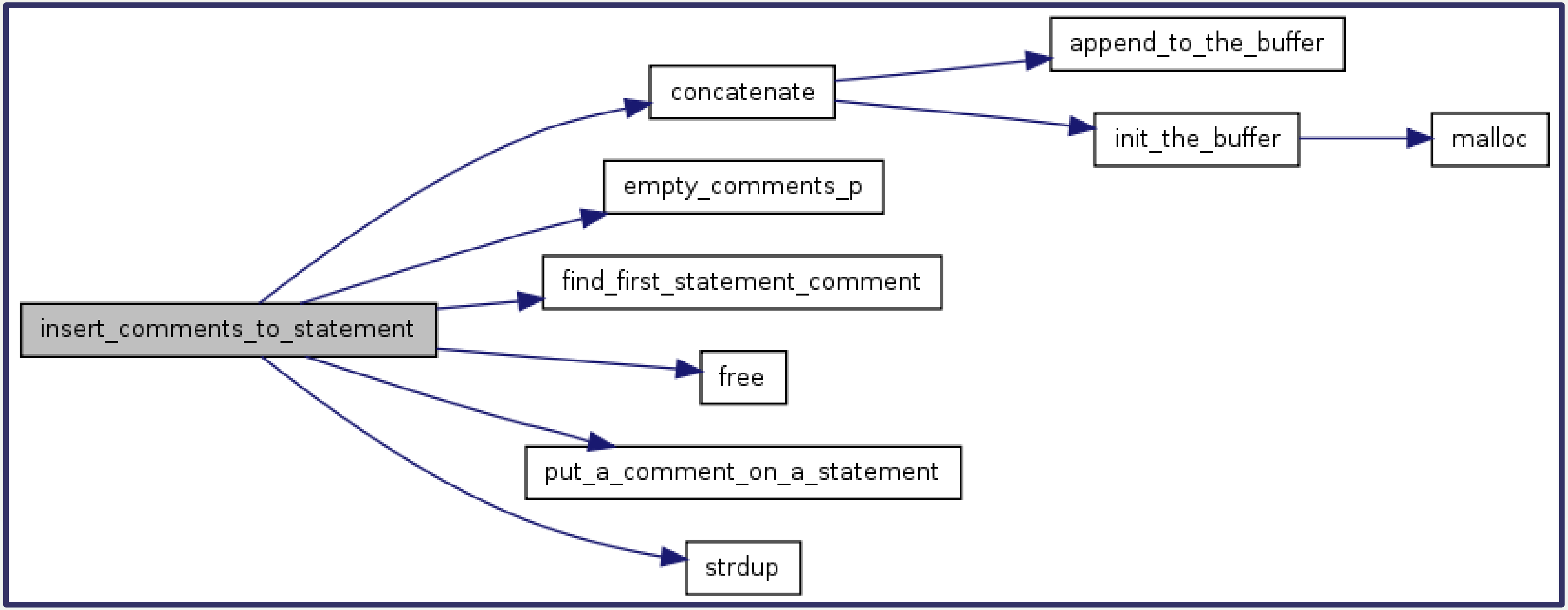
```
$ grep comment TAGS | grep statement | sort -u | grep -v "#define" | grep -v static
append_comments_to_statement(1413,36294
bool empty_statement_or_continue_without_comment_p(357,8436
discard_statement_and_save_label_and_comment(191,5841
find_first_statement_comment(1328,33859
gather_all_comments_of_a_statement(1317,33560
gather_all_comments_of_a_statement_filter(1296,32940
insert_comments_to_statement(1439,36891
list get_statements_with_comments_containing 491,13641
put_a_comment_on_a_statement(1387,35547
set_comment_of_statement(760,17551
statement add_comment_and_line_number(1504,38475
3,11332
```

```
/* Insert a comment string (if non empty) at the beginning of the
   comments of a statement.

   @param c is strdup'ed in this function.
*/
void insert_comments_to_statement(statement s, string c)
```

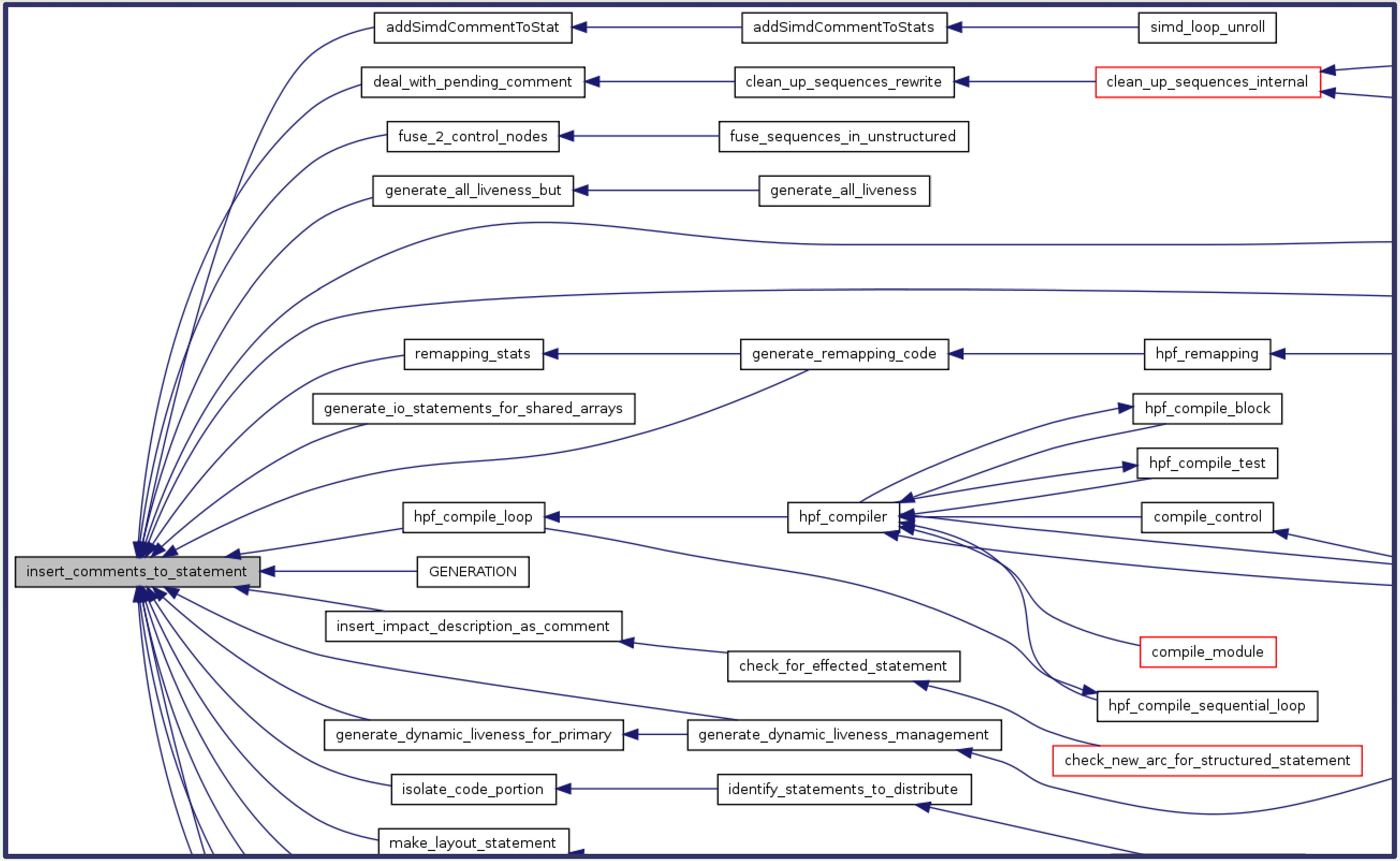
Browsing the Doxygen Documentation



Browsing the Doxygen Documentation: Callees



Browsing the Doxygen Documentation: Callers



Fill in the New C File (cont.)

- **Step 5: find the proper IR function to read a PIPS property**

```
bool prepend_comment(string mn) {

  // Use this module name to get the resources we need
  statement s = PIPS_PHASE_PRELUDE(mn,
                                "PREPEND_COMMENT_DEBUG_LEVEL" );

  // Get the value of the prope
  string c = ... ;

  // Add comment c to the mo
  s = ... ;

  // Put back the new stateme

  PIPS_PHASE_POSTLUDE
}
```

Check libraries
Check Doxygen modules

```
$ grep property $PIPS_ROOT/src/Libs/properties/properties.h
#include "property.h"
#define pips_flag_p(p) get_bool_property(p)
#define pips_flag_set(p) set_bool_property((p), TRUE)
#define pips_flag_reset(p) set_bool_property((p), FALSE)
extern bool get_bool_property(const string /*name*/);
extern void set_bool_property(const string /*name*/, bool /*b*/);
extern string get_string_property(const string /*name*/);
extern string get_string_property_or_ask(const string /*name*/, const char
/*question*/[]);
extern void set_string_property(const string /*name*/, string /*s*/);
extern int get_int_property(const string /*name*/);
extern void set_int_property(const string /*name*/, int /*i*/);
extern void fprint_property_direct(FILE */*fd*/, const string /*pname*/);
extern void fprint_property(FILE */*fd*/, const string /*pname*/);
```

See Pipsmake Documentation

The Final C File

- **Step 6: put the C file together**

```
#include "genC.h"
#include "linear.h"
#include "ri.h"
#include "ri-util.h"
#include "misc.h"
#include "pipsdbm.h"
#include "resources.h"

bool prepend_comment(string mn) {

  statement s = PIPS_PHASE_PRELUDE(
    mn, "PREPEND_COMMENT_DEBUG_LEVEL");

  string c = get_string_property("PREPEND_COMMENT");

  insert_comments_to_statement(s, c);

  PIPS_PHASE_POSTLUDE(s);
}
```

Declare the New Phase in PIPS framework

- **File:** $PIPS_ROOT/src/Documentation/pipsmake/pipsmake-rc.tex

```
\section{Prepending a comment}
\label{sec:prepending-comment}

\begin{PipsPass}{prepend_comment}
Prepends a comment to the first statement of a module.
Useful to apply post-processing after PIPS.
\end{PipsPass}

\begin{PipsMake}
alias prepend_comment 'Prepend a comment to the first statement of a module'

prepend_comment > MODULE.code
  < PROGRAM.entities
  < MODULE.code
\end{PipsMake}

The comment to add is defined by this property:
\begin{PipsProp}{PREPEND_COMMENT}
PREPEND_COMMENT "/* This comment is added by PREPEND_COMMENT phase */"
\end{PipsProp}
```

This is LaTeX!

Default property value

Integrate the New Phase: Unbuild & Build

- **Rebuild PIPS:**

```
$ cd $PIPS_ROOT
$ make unbuild      # stronger than "make clean"
$ make -j build     # to compile the source and regenerate the documentation
```

- **Some outputs of pipsmake-rc.tex:**

Literate Programming!

  - **properties.rc**: the options and their default values
  - **pipsmake.rc**: the rules linking phases and resources
  - **resources.h**: the names of the resources
  - **phases.h**: the names of the phases
  - **builder_map.h**: the links between phase names and C functions
  - Some menus and documentation for PIPS user interfaces
  - And the user documentation, **pipsmake-rc.pdf** and **pipsmake-rc.html**

Extend the Tpips Script

- Apply this transformation to all modules:

  %ALLFUNC

- Get the modified source file back:

  UNSPLIT

```
delete extending01b
create extending01b extending01b.c

setproperty PREPEND_COMMENT " PPoPP Bangalore"

apply PREPEND_COMMENT[%ALLFUNC]

apply UNSPLIT[extending01b]

close
quit
```

```
/* file for extending01b.c */
extern float extending01a(int n, float a[n]);
extern float extending01b(int n, float a[n]);

float extending01a(int n, float a[n])
{
   // PPoPP Bangalore
   int i;
   float s = 0.;
   for(i = 0; i <= n-1; i += 1)
      s += a[i]*a[i];
   return s;
}
float extending01b(int n, float a[n])
{
   // PPoPP Bangalore
   int i;
   float s = 0.;
   for(i = 0; i <= n-1; i += 1)
      s += a[i]*a[i];
   return s;
}
```

Wrap-up: Finding the Proper IR Function and more...

- **PIPS libraries**
  - **General structure** (see Slide III.1.5)
  - **Doxygen**: http://doxygen.pips.enstb.org/PIPS/graph
  - **TAGS** (see Slide III.1.9)
  - **Library headers** (automatically generated by cproto)
    - Do not update them directly...
  - Google and Google Code: http://google.com/codesearch
  - pipsdev-at-cri.mines-paristech.fr **mailing-list**
  - **IRC**: irc://irc.freenode.net/pips
- **Use...**
  1. **properties** to pass parameters (and possibly return results)
  2. **pipsmake-rc.tex** to declare phases, resources and properties: they become available for all PIPS user interfaces
  3. the **tpips script interface** to check the result and to define non-regression test cases

Extending PIPS: Example 2

Extending PIPS - Example 2

Example 2: Add a Call Site

III.2.1

- **Purpose: track function activations**
  - Add a **call to a run-time function** as **first statement of a module**
  - Assume the **source code** of the run-time function is **not available**
    - Maybe it's assembly code...
  - The function is a **global function**, with a specific name, MY_TRACK
    - A **property** could be used to define the name of the function
  - The function MY_TRACK
    - does not have any argument
    - and does not return any value

```
float extending02(int n, float a[n])
{
  int i;
  float s=0.;
  for(i=0;i<n;i++) {
    s += a[i]*a[i];
  }
  return s;
}
```

```
delete extending02
create extending02 extending02.c

apply PREPEND_CALL
display PRINTED_FILE

close
quit
```

```
float extending02(int n, float a[n])
{
   int i;
   float s = 0.;
   MY_TRACK();
   for(i = 0; i <= n-1; i += 1)
      s += a[i]*a[i];
   return s;
}
```

What do I need to do?

III.2.2

- **What I've got from Example 1:**
  - I've got my copy of PIPS, in **`$PIPS_ROOT`**
  - I've got my new library, **`to_begin_with`**
  - I've got a C file, **`add_stuff_to_module.c`**

  → Let's reuse them!
- **New stuff I'll need:**
  - I need to make a **function entity** with:
    - A name
    - A type
    - An initial value
    - A storage
  - Make a **call site**
  - **Make** a statement
  - **Insert** the statement

All these objects are managed by Newgen

ri-util handles this for us

Internal Representation: Newgen declarations

Excerpt from $PIPS_ROOT/src/Documentation/newgen/ri.newgen :

- **statement** = label:entity
  x number:int x ordering:int
  x comments:string
  x **instruction**
  x declarations:entity*
  x decls_text:string x extensions;
- **instruction** = sequence + test
  + loop + whileloop
  + goto:statement
  + **call**
  + unstructured + multitest
  + forloop + expression;
- **call** = function:entity
  x arguments:expression*;

- **Newgen syntax:**
  - x : build a **structure**
  - + : build a **union**
  - * : build a **list**
  - string, int, float, …: **basic types**
  - Also set **{}**, array **[]** and mapping **->**
- **Documentation:**
  - http://pips4u.org/doc/manuals
    (ri.pdf, ri_C.pdf)

In French: *Représentation Interne*, hence the many “ri”

The Symbol Table

Entity?

- Variables
- Functions
- Intrinsics
- Constants
- Values
- ...

- tabulated **entity** = name:string x **type** x **storage** x initial:**value**
- **type** = statement:unit + area + variable + **functional** + varargs:type + unknown:unit + void:unit + struct:entity* + union:entity* + enum:entity*
  - type rt = make_type_void();
  - type t = make_type_functional(f);
- **value** = **code** + symbolic + constant + intrinsic:unit + unknown:unit + expression
  - value v = make_value_code(c);
- **functional** = parameters:parameter* x result:type
  - functional f = make_functional(NIL, rt);
- **code** = declarations:entity* x decls_text:string x initializations:sequence x externs:entity* x language
  - code c = make_code(NIL, empty_string, make_sequence(NIL), NIL, make_language_c());
- **storage** = return:entity + ram + formal + rom:unit

→ **Newgen generates basic functions**

- factories, accessors, iterators,...

The Internal Representation Library: ri-util

- **35 KLoC…**

Automatically generated header file

Header file to update

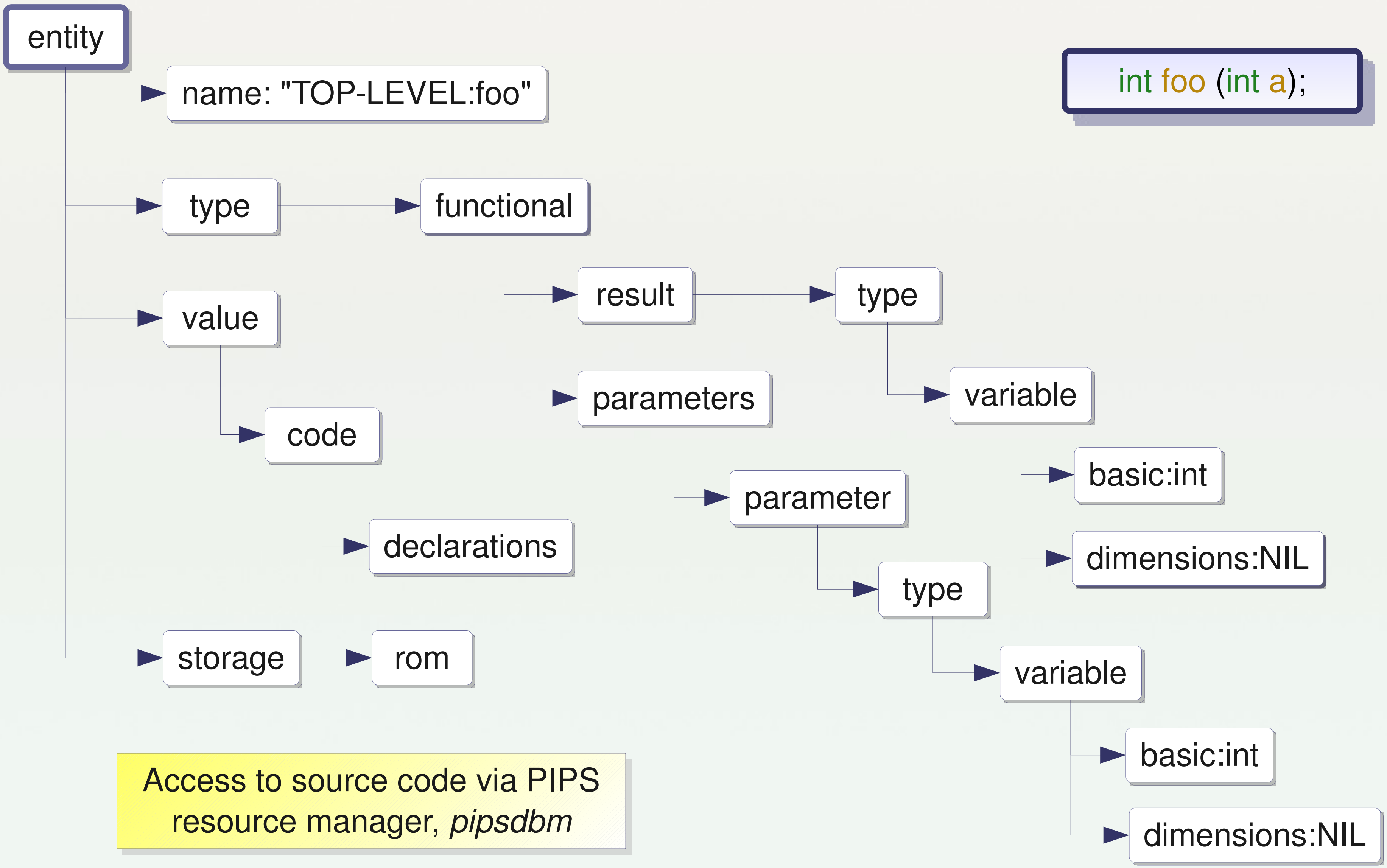
```
$ ls $PIPS_ROOT/src/Libs/ri-util
arguments.c                  craft.c          loop.c             ri-util.h
attachment_pretty_print.c    declarations.c   Makefile           ri-util-local.h
bound_generation.c           effects.c        misc_paf_utils.c   size.c
clean.c                      entity.c         module.c           statement.c
clone_statement.c            eval.c           normalize.c        static.c
cmfortran.c                  expression.c     operator.h         type.c
constant.c                   fortran90.c      ordering.c         unstructured.c
contrainte_to_text.c         hpfc.c           pragma.c           util.c
control.c                    instruction.c    prettyprint.c      variable.c
convex_hull.c                LINUX_x86_64_LL  replace.c
```

$PIPS_ARCH

AST
Key internal representation components

Not *designed*, but *accumulated*… (ongoing refactoring)

Internal Representation: A Function



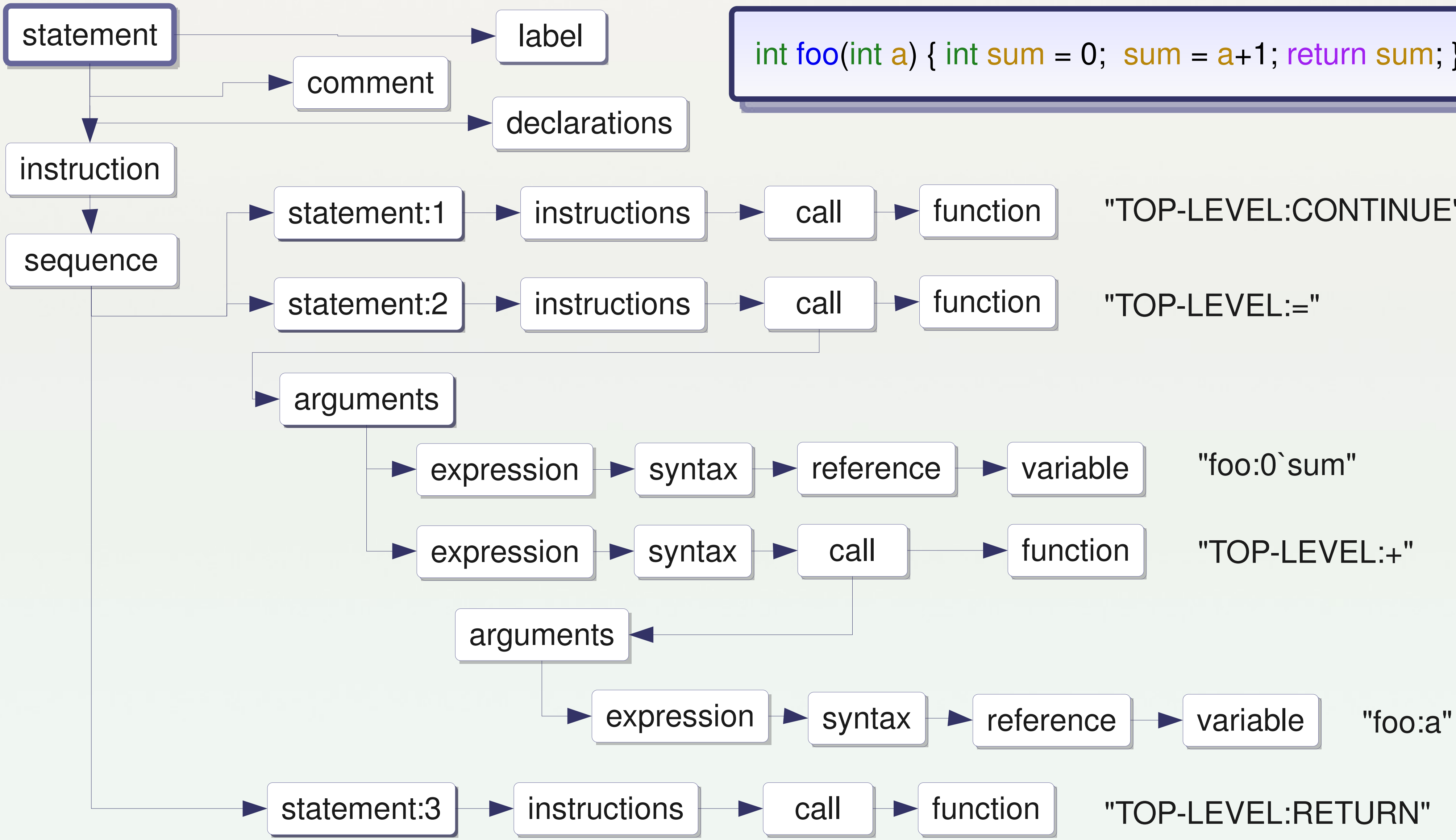

```
int foo (int a);
```

Access to source code via PIPS resource manager, *pipsdbm*

Internal Representation: A Statement

III.2.7

```
int foo(int a) { int sum = 0;  sum = a+1; return sum; }
```



The Internal Representation Interface: ri

III.2.8

- **Excerpt from** $PIPS_ROOT/include/ri.h:

```
#define statement_undefined ((statement)gen_chunk_undefined)
#define statement_undefined_p(x) ((x)==statement_undefined)
```

Automatically generated by Newgen

```
extern statement make_statement(entity, intptr_t, intptr_t, string, instruction, list, string, extensions);
extern statement copy_statement(statement);
extern void free_statement(statement);
```

Memory management

```
extern statement check_statement(statement);
extern bool statement_consistent_p(statement);
extern bool statement_defined_p(statement);
```

Debugging
Dynamic type checking

```
extern list gen_statement_cons(statement, list);
```

Typed lists

```
extern void write_statement(FILE*, statement);
extern statement read_statement(FILE*);
```

ASCII Serialization

```
// gen_context_multi_recurse(obj, context, [domain, filter, rewrite,] * NULL);
```

Iterators

Writing the C Code


```
#include "newgen.h"
#include "linear.h"
#include "ri.h"

bool prepend_call(string mn) {
  type rt = make_type_void();
  functional ft = make_functional(NIL, rt);
  type t = make_type_functional(ft);
  storage st = make_storage_rom();
  code co = make_code(NIL, strdup(""), make_sequence(NIL), NIL,
                      make_language_unknown());
  value v = make_value_code(co);
  string ffn = strdup(concatenate(TOP_LEVEL_MODULE_NAME,
                                  MODULE_SEP_STRING, "MY_TRACK", NULL));
  entity f = make_entity(ffn, t, st, v); call ca = make_call(f, NIL);
  instruction i = make_instruction_call(ca);
  statement s = instruction_to_statement(i);
  statement module_statement = PIPS_PHASE_PRELUDE(mn, "PREPEND_CALL_DEBUG_LEVEL");
  insert_statement(module_statement, s, TRUE);
  PIPS_PHASE_POSTLUDE(module_statement);
}
```

Pedagogical only!
Illustration of Newgen style.
This kind of code is available in library ri-util.

Don't forget to...

- Update pipsmake-rc.tex (see Slide III.1.15)
- Rebuild PIPS: **`make unbuild && make build`** (see Slide III.1.16)
- Create a test case:

```
float extending02(int n, float a[n])
{
  int i;
  float s=0.;
  for(i=0;i<n;i++) {
    s += a[i]*a[i];
  }
  return s;
}
```

```
delete extending02
create extending02 extending02.c

apply PREPEND_CALL
display PRINTED_FILE

close
quit
```

```
float extending02(int n, float a[n])
{
   int i;
   float s = 0.;
   MY_TRACK();
   for(i = 0; i <= n-1; i += 1)
      s += a[i]*a[i];
   return s;
}
```

Oops... Missing Source Code

- **Display call graph:**

```
delete extending02b
create extending02b extending02.c

apply PREPEND_CALL
display PRINTED_FILE
display CALLGRAPH_FILE[extending02]
close
quit
```

```
Request: build resource CALLGRAPH_FILE for
user warning in build_real_resources: No source code for module MY_TRACK.
  C_INITIALIZER      building   C_SOURCE_FILE(MY_TRACK)
                     building   C_SOURCE_FILE(MY_TRACK!)
user error in generic_initializer: no source file for MY_TRACK (might be an ENTRY point)
set PREPROCESSOR_MISSING_FILE_HANDLING to "query" or "generate"...
user warning in set_debug_stack_pointer: debug level stack is set to 2
user error in build_real_resources: unable to build callees for MY_TRACK
Some source code probably is missing!
```

- **Request generation of missing source code:**

```
setproperty PREPROCESSOR_MISSING_FILE_HANDLING "generate"
```

```
CALLGRAPH_FILE made for extending02.
 extending02
     MY_TRACK
```

Wrap-up of Example 2

- **Newgen as STL**
- **Internal Representation, including the symbol table**
- **Library ri-util + newgen run-time**
- **Missing source code management**
- **Non-regression test cases**

Example 3: Array Scalarization

Extending PIPS - Example 3: Array Scalarization

Example 3: Array Scalarization

- **What do I want to do?**
  - Within a loop, replace all references x[i][j] to an array x by references to a scalar
- **It maybe useful after a loop fusion**
- **When can it be done?**
  - Each iteration should access only one array element
- **When does it seem to be useful?**
  - Copy-in overhead? Copy-out overhead?

Read/Write regions of the loop body

IN regions...

OUT regions...

```
subroutine scalarization(x,y,n)
real x(n,n), y(n,n), t(100)
do i = 1,n
  do j = 1,n
    t(i)   = x(i,j)
    x(i,j) = y(i,j)
    y(i,j) = t(i)
  enddo
enddo
end
```

```
SUBROUTINE SCALARIZATION(X,Y,N)
REAL X(N,N), Y(N,N), T(100)
DO I = 1, N
  DO J = 1, N
    __scalar__0 = X(I,J)
    X(I,J) = Y(I,J)
    Y(I,J) = __scalar__0
  ENDDO
ENDDO
END
```

Working with SVN Branches in PIPS

- **Why get an SVN branch?**
  - Add versioning to your developments
  - Possibility of later merging your developments into the PIPS SVN trunk
- **How do I start working with my SVN Branch?**
  - Request a **SVN developer account** from CRI — Ask us!

    → login "calvin" + directory "/branches/calvin" in SVN repository
  - Get a **working copy** of your dev directory:

    ```
    $ cd MYPIPS
    $ svn co http://svn.cri.mines-paristech.fr/svn/pips/branches/calvin pips_dev
    $ cd pips_dev
    ```
  - Create a **dev branch** from trunk:

    ```
    $ svn cp http://svn.cri.mines-paristech.fr/svn/pips/trunk scalarization
    $ svn ci
    $ cd scalarization
    $ make build
    ```
  - **Edit + tests + svn ci** development cycle

Updating my Working Copy from the Trunk

Check out the latest changes from the SVN trunk:

1. I don't have a branch, I'm working in **MYPIPS/prod/pips**:

   $ cd MYPIPS/prod/pips
   $ **svn up** # this line does the update!
   $ make unbuild
   $ make build

What if **Linear** or **Newgen** need rebuilding, too?
→ build from MYPIPS/prod!

2. I'm « working in my branch », **MYPIPS/pips_dev/scalarization**:

   $ cd MYPIPS/pips_dev/scalarization
   $ **svn up** # this only updates the **revision number** of my working copy!
   $ **svn merge http://svn.cri.ensmp.fr/svn/pips/trunk** # actual update is done here

   # Conflicts?
   $ svn ci # this commits updates into my SVN branch
   $ make unbuild
   $ make build

Use SVN **automatic completion**!

- **Reminder**:

  « *working in my branch* » really means « *using a **working copy** of my SVN branch* »

Getting and Setting PIPS Resources

- **Convex array regions: IN, OUT, READ/WRITE**
  - READ/WRITE: check that only one element is used per iteration
  - IN and OUT: detect if copy-in and/or copy-out code is necessary
- **Regions are functions from the store to convex sets of array elements: see Slide II.1.7**

Math!

- **They are "resources" for pipsmake and pipsdbm**
- **Simply request them in pipsmake-rc.tex (see Slide III.1.15):**

```
scalarization  > MODULE.code
        < PROGRAM.entities
        < MODULE.code
        < MODULE.regions
        < MODULE.in_regions
        < MODULE.out_regions
```

Global Resource Coherence: *pipsmake* and *pipsdbm*

- **Ensure that necessary resources are available and up-to-date:**
  - They are declared in pipsmake-rc.tex,
  - *Pipsmake* computes them in the right order,
  - Each phase gets them when needed.
- **Logical timestamps are used:**
  - to detect available up-to-date ressources
  - to recompute the other resources transparently.
- **Logical time incremented each time a resource is stored in *pipsdbm*.**
- **The current workspace contains all resources related to an application.**
- **The workspaces can be closed and re-opened**
  - **Persistence** managed by ***pipsdbm***.
  - ASCII serialization

PIPS Resource Management: the Workspace

WORKSPACE: my_ws.database

MODULE 1: intro01!
CALLEES CODE C_SOURCE_FILE DECLARATIONS
PARSED_CODE PRINTED_FILE USER_FILE
intro01!.c intro01!.pre.c

MODULE 2: main
CALLEES CODE C_SOURCE_FILE
PARSED_CODE PRINTED_FILE USER_FILE
main.c main.pre.c

Metadata
pipsmake properties ...

Program
ENTITIES Program.txt USER_FILE

Tmp
intro01.cpp_processed.c ...

Src
intro01.c

Logfile

Warnings

create my_ws intro01.c
apply UNSPLIT

Compilation Unit

Resource DBR_CODE

db_get_memory_resource(DBR_CODE, module_name, TRUE)

Function

Symbol Table

Transformed version of the input file

Iterating across PIPS AST: gen_recurse

- **How to substitute all scalarized references in the loop body?**

```
static void statement_substitute_scalarized_array_references(statement st, entity a, entity s)
{
  scalarized_array = a;
  scalarized_replacement_variable = s;
  gen_recurse (st, reference_domain, reference_substitute, gen_null);
  scalarized_array = entity_undefined;
  scalarized_replacement_variable = entity_undefined;
}
```

Replace references to array a by references to scalar s in statement st

Lookup all references in statement st and call reference_substitute

Downward traversal and update...

```
static bool reference_substitute(reference r) {
  bool result = FALSE; entity v = reference_variable(r);
  if (v == scalarized_array) {
    list inds = reference_indices(r);
    size_t d = type_depth(ultimate_type(entity_type(v)));
    if (gen_length(inds) == d) {
      reference_variable(r) = scalarized_replacement_variable;
      reference_indices(r) = NIL;
      result = TRUE; } }
  return result; }
```

If reference r is a subscripted reference to a

Update reference r and its subscripts

Memory Leak?

gen_context_multi_recurse()

- **Let's avoid global variables:**
  - substitute all use references to v by expression e: reference_rewrite()
  - remove all def references to v: filter_assignment()
  - perform the update during the **upward phase**

```
struct substitution {
  entity v;
  expression e ;
} sub;

gen_context_multi_recurse(s, sub,
                          call_domain, filter_assignment, gen_null,
                          reference_domain, gen_true, reference_rewrite,
                          NULL);
```

- bool filter_assignment(call ca, void * c)
- void reference_rewrite(reference r, void * c)

Using PIPS Proof Engine: Linear Library

- **Sparse Representation**
- **Polyhedral libraries**
  - Library hierarchy:
    - arithmetique, vecteur, contrainte, sc,
    - ray_dte, sommet, sg,
    - matrice, matrix, plint,
    - Polyedre, union, polynome
  - Automatic generation of header files
  - Standalone
- **Overflow control inside**
- **Presburger arithmetic with union, list of polyhedra**
- **How to find the function I need?**
  - TAGS in $PIPS_ROOT/../linear/TAGS
  - In what Linear sub-library should it be? Does it need the dual representation?

Used by Polylib

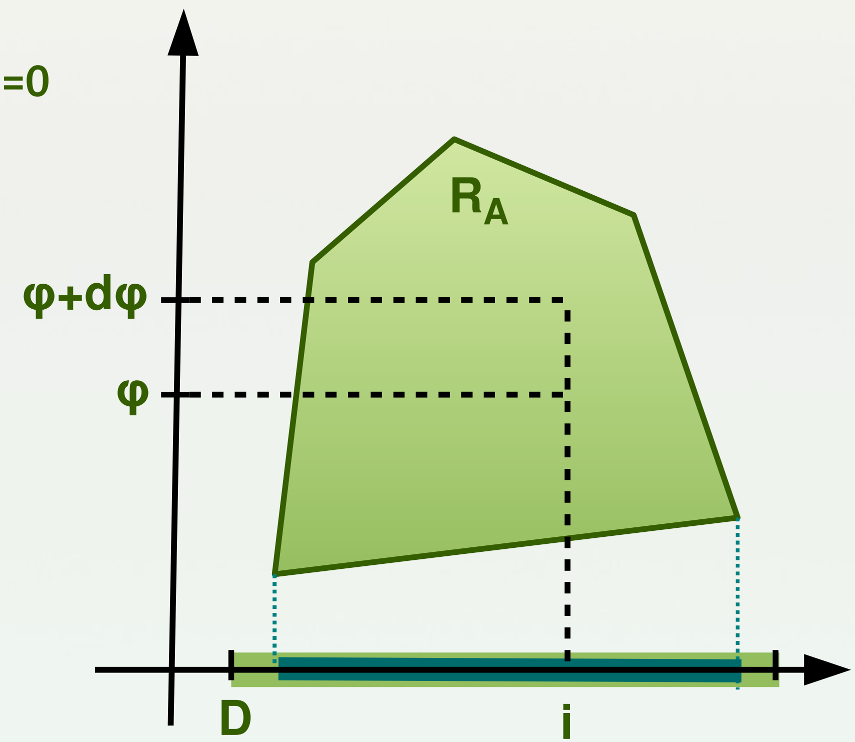
1986 vintage code

Linear algebra?
Extra-library: polynome

Or use Eclipse...

Proving with Linear

- Check that a unique element φ of the array **A** is used at each iteration **i**.
  - i.e. the relation R_A between the iterations **i** and the array element φ is a function
- The proof is based on
  - $r_A(i)=\varphi \wedge r_A(i)=\varphi+d\varphi \wedge$ "r is a function" $\Rightarrow d\varphi=0$
  - If $\exists\ d\varphi \neq 0$, then r_A cannot be a function.
- So:
  - we build $R_A(i, \varphi)$ and $R_A(i, \varphi+d\varphi)$,
  - project their conjunction on the $d\varphi$ subspace,
  - and check that $d\varphi = 0$



- And each iteration **i** must use at least one element φ.
  - i.e. the function r_A must be a total function of **i**.
- The projection of the graph R_A on the domain must include the iteration set **D** of **i**.

Coding with Linear

- **Check that a relation graph is a total function graph**

```
bool sc_totally_functional_graph_p( Psysteme g, // function graph
                                   Pbase d,      // domain's basis
                                   Psysteme D, // membership predicate for functional domain
                                   Pbase r,      // range's basis
                                   Pbase dr      // difference variables
                                   )
{
  bool totally_functional_p = FALSE;
  if (sc_functional_graph_p(g, d, r, dr)) {
    // Check args coherence: d should be included in D's basis.
    if (base_included_p(d, sc_base(D))) {
      // Project graph g on domain space d along the range basis r.
      Psysteme g1 = sc_copy(g);
      sc_projection_along_variables_ofl_ctrl(&g1, r, OFL_CTRL);
      // By definition of a total function, D must be included in g1.
      totally_functional_p = sc_inclusion_p_ofl_ctrl(D, g1, OFL_CTRL);
    }
  }
  return totally_functional_p;
}
```

Overflow control!

Debugging Newgen data structures

- **gdb (or better, debug under Emacs or Eclipse):**

```
$ gdb ../../prod/pips/bin/LINUX_x86_64_LL/tpips
(gdb) break reference_substitute
Breakpoint 1 at 0x4a792c: file $PIPS_ROOT/src/Libs/transformations/scalarization.c, line 234.
(gdb) r extending03.tpips

user warning in loop_scalarization: Creating variable SCALARIZATION:__scalar__0 for variable
SCALARIZATION:T

Breakpoint 1, reference_substitute (r=0xcaa180)
    at /home/francois/MYPIPS/prod/pips/src/Libs/transformations/scalarization.c:234
234     bool result = FALSE;
(gdb) print *r
$1 = {_type_ = 75, _reference_variable_ = 0xc9eeb0, _reference_indices_ = 0x0}
(gdb) print *r->_reference_variable_
$2 = {_type_ = 84, _entity_index__ = 590, _entity_name_ = 0xc9a950 "SCALARIZATION:N",
  _entity_type_ = 0xca7540, _entity_storage_ = 0xca7580, _entity_initial_ = 0xca75a0}
```

Dynamic Newgen type

Scalar reference

Non-Regression Tests

- **Non-regression tests are clustered library by library**
- **They are managed with SVN**
  - setup_pips.sh checks a working copy out
- **Each test case includes:**
  - A source file or a set of source files
  - A tpips script
  - A result directory with the expected result file, test
- **For instance:**

```
~/MYPIPS/validation/TutorialPPoPP2010$ ls extending03.*
extending03.f  extending03.tpips

extending03.result:
test
```

- **A *default_tpips* file may be defined in the library test directory**
- $ make validate

Sharing my New Transformation

- **My new transformation is ready, I'd like to send it to the PIPS SVN trunk** so that other people can use it.
  - I don't use a branch, I'm working in **MYPIPS/prod/pips**:

    $ cd MYPIPS/prod/pips

    $ **svn ci # Caution!!! This line actually updates the SVN trunk!**
  - I'm working in my branch, **MYPIPS/pips_dev**:

    $ **cd MYPIPS/prod/pips** # Go to the working copy of the trunk

    $ **svn up** # this ensures my working copy of the trunk is up to date

    $ **svn merge --reintegrate **

    http://svn.cri.ensmp.fr/svn/pips/branches/calvin/scalarization

    # Check conflicts, unbuild+build, make validate...

    $ svn ci # Commits the merged version into the SVN trunk. Caution!!!

Which means I'm a **core developer** with **full committer's rights**!

Wrap-Up: PIPS Architecture

- **PIPS framework: easy to extend with some LaTeX declarations**
- **Coherence: *pipsmake***
  - Takes care of interprocedural issues,
  - Automatically provides the requested resources
- **Persistence: *pipsdbm***
  - Can be useful to analyze large applications (checkpoint/restart)
  - Can be useful to exploit PIPS results without linking with PIPS
- **External libraries used:**
  - Newgen, key for would-be developpers
  - Linear, Polylib, readline...
- **Non-regression tests: validation**
- **SVN environment, Trac reporting & code browsing**
- **Online documentation:** http://pips4u.org/doc

Wrap-Up: PIPS Internal Representation (IR)

- **Based on Newgen**
- **Introduction to Newgen**
- **Layered “classes” using the “persistent” keyword**
- **Iterators: gen_recurse(), gen_context_multi_recurse()**
- **Access to Symbol Table**
- **Library ri-util**
  - in French, *Utilities for the Internal Representation*

IV. A Python PIPS API

IV. A Python PIPS API

IV. A Python PIPS API

- **Goals:**
  - Make tpips more flexible (python > shell?)
  - Develop generic tpips (no hard-coded values)
  - Easier high-level extensions to PIPS using high-level python modules
- **Why Python?**
  - Scripting language
  - Natural syntax
  - Easy C binding using swig
- **Be nice with new developers! (Plenty of pythonic tasks)**
  - ipython integration
  - Python-gtk binding
- **Attract (lure?) users!**
  - Combine transformations easily
  - Develop high-level tools based on PIPS

From Tpips to Pyps

IV.1.1

```
create $WKS jacobi.c p4a_stubs.c

setproperty LOOP_NORMALIZE_ONE_INCREMENT TRUE
setproperty LOOP_NORMALIZE_LOWER_BOUND 0
setproperty LOOP_NORMALIZE_SKIP_INDEX_SIDE_EFFECT TRUE
apply LOOP_NORMALIZE[%ALLFUNC]

apply PRIVATIZE_MODULE[%ALLFUNC]

activate PRINT_CODE_REGIONS
display PRINTED_FILE[%ALLFUNC]

activate PRINT_CODE
apply COARSE_GRAIN_PARALLELIZATION[%ALLFUNC]
display PRINTED_FILE[compute]

apply GPU_IFY[%ALLFUNC]
apply KERNEL_LOAD_STORE[p4a_kernel_launcher_0,
p4a_kernel_launcher_1, p4a_kernel_launcher_2,
p4a_kernel_launcher_3, p4a_kernel_launcher_4]
apply GPU_LOOP_NEST_ANNOTATE[p4a_kernel_launcher_0,
p4a_kernel_launcher_1, p4a_kernel_launcher_2,
p4a_kernel_launcher_3, p4a_kernel_launcher_4]

apply INLINING[p4a_kernel_0, p4a_kernel_1, p4a_kernel_2,
p4a_kernel_3, p4a_kernel_4]
...
```

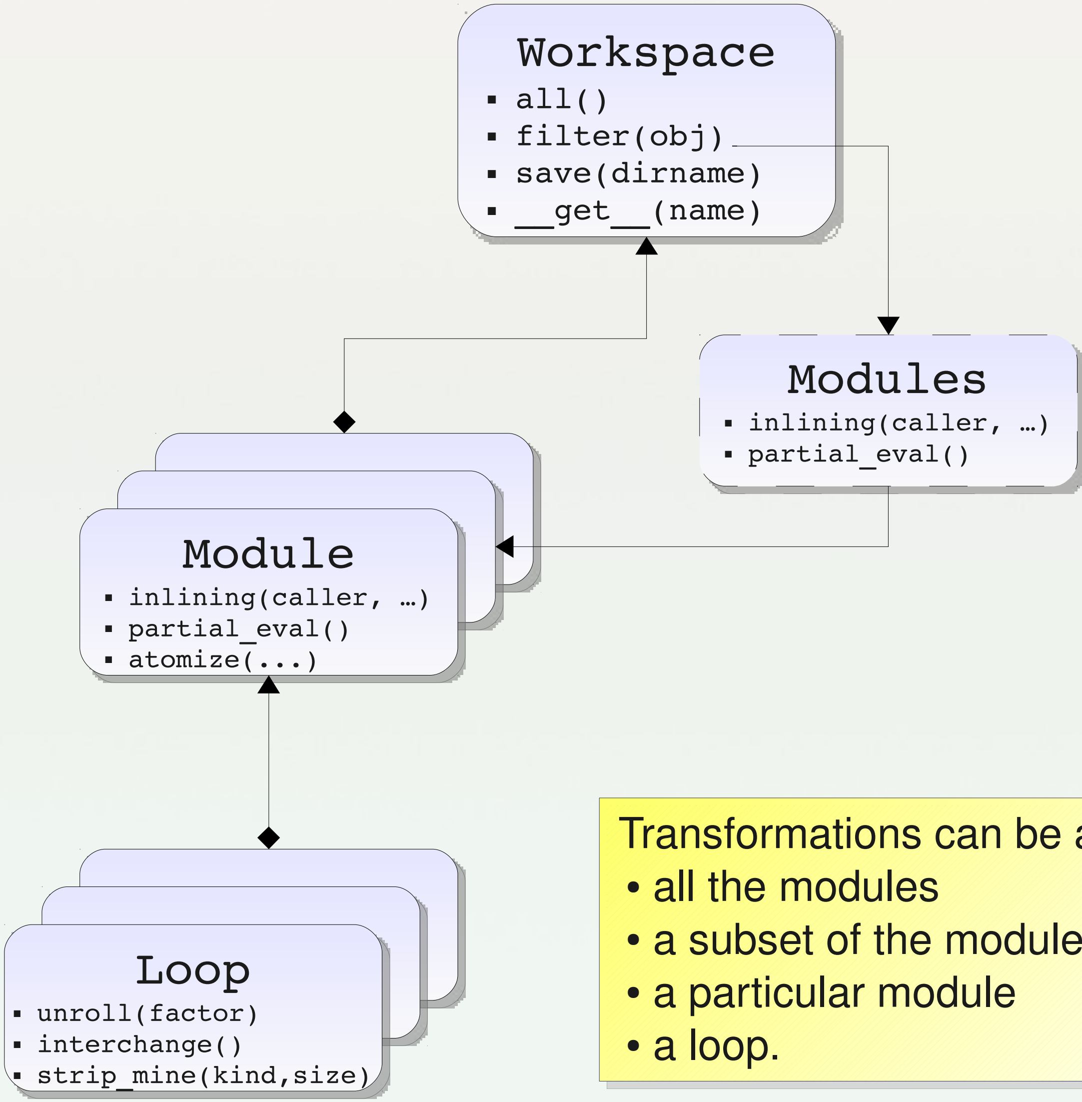
```
from pyps import *

w = workspace(['jacobi.c','p4a_stubs.c'])

w.set_property(loop_normalize_one_increment = True,
          loop_normalize_lower_bound=0,
          loop_normalize_skip_index_side_effect=True)

w.all.loop_normalize()
w.all.privatize_module()

w.all.display(With='PRINT_CODE_REGIONS')

w.all.coarse_grain_parallelization()
w.all.display()

w.all.gpu_ify()

launchers= modules(['p4a_kernel_launcher_0',
'p4a_kernel_launcher_1', 'p4a_kernel_launcher_2',
'p4a_kernel_launcher_3', 'p4a_kernel_launcher_4'])
launchers.kernel_load_store()
launchers.gpu_loop_nest_annotate()
launchers.inlining()
...
```

Pyps Example


```
from pyps import *

w = workspace(['jacobi.c','p4a_stubs.c'])

w.set_property(loop_normalize_one_increment = True,
        loop_normalize_lower_bound=0,
        loop_normalize_skip_index_side_effect=True)

w.all.loop_normalize()
w.all.privatize_module()

w.all.display(With='PRINT_CODE_REGIONS')

w.all.coarse_grain_parallelization()
w.all.display()

w.all.gpu_ify()

launchers= modules(['p4a_kernel_launcher_0',
'p4a_kernel_launcher_1', 'p4a_kernel_launcher_2',
'p4a_kernel_launcher_3', 'p4a_kernel_launcher_4'])
launchers.kernel_load_store()
launchers.gpu_loop_nest_annotate()
launchers.inlining()
...
```

```
from pyps import *
import re

launcher_re = re.compile("^p4a_kernel_launcher_.*")
def launcher_filter(module):
        return launcher_re.match(module.name)

w = workspace(["jacobi.c","p4a_stubs.c"])

w.all.loop_normalize(one_increment=True,lower_bound=0,skip_index_s
ide_effect=True)
w.all.privatize_module()

w.all.display(With="PRINT_CODE_REGIONS")

w.all.coarse_grain_parallelization()
w.all.display()

w.all.gpu_ify()

# select only some modules from the workspace
launchers=w.all(launcher_filter)
# manipulate them as first level objects
launchers.kernel_load_store()
launchers.display()

launchers.gpu_loop_nest_annotate()
launchers.inlining()
...
```

Pyps Class Hierarchy



- Programs, Modules and Loops are first-level objects
- Collection of modules have the same interface as single modules

- Transformation extension through inheritance
- Transformation chaining with new methods
- Workspace hook through inheritance

Transformations can be applied to:
- all the modules
- a subset of the modules,
- a particular module
- a loop.

```
$ sudo apt-get install python-pips
$ pydoc pyps
```

An Example of Application: Genetic Algorithm

- **Goal:**
  - "Transformation space exploration": find a good transformation set for a given application
- **How:**
  - Explore the possibilities using a genetic algorithm
  - Use pyps to dynamically
    - create workspaces
    - apply transformation sets
    - generate new source files
    - benchmark them
- **Extensions:**
  - Use it as a "fuzzer"
  - Use "Pyro" for distributed exploration

Developed
in partnership
with

An Example of Application: *pipscc*

- ***Pipscc*:**
  - A wrapper that behaves like a standard compiler but applies source-to-source interprocedural transformations before the compilation.

    ```
    $ pipscc foo.c ; ./a.out
    $ pipscc -c foo.c ; pipscc -c bar.c ; pipscc foo.o bar.o -o foobar
    ```
  - *Pipscc* is a nice building block for assembling high-level tools using PIPS.
  - Less than 200 lines of Python
- **Implementation:**
  - `*.c (gcc -c)→ *.o (gcc)→ a.out`
  - `*.c (pipscc -c)[1] → *.o (pipscc)[2] → a.out`
    - [1]: `cpp` + pickle (object serialization)
    - [2]: unpickle + transformations + compilation + link

Example: Selective Inlining with *pipscc*

- **Goal:**
  - Inline functions with fewer than 3 lines of code
- **How:**
  - Create a foo.py file which defines a subclass with a single method "changes"
  - `# CC="python foo.py"`
- **Notes:**
  - "changes" could be a call to a third party-library, another source code analyzer …
  - Python provides bindings from various languages

```
import pyps
from pipscc import pipscc

class Pii(pipscc):

    def changes(self, ws):

        def thefilter(module):
            return len(module.code()) < 3

        ws.filter(thefilter).inlining()


if __name__ == '__main__':

    thecompiler = Pii()
    thecompiler.run()
```

V. Ongoing Projects Based on PIPS

V.0.1

V. Ongoing Projects Based on PIPS

V. Ongoing Projects Based on PIPS

V.0.2

- **What can you do by combining basic analyses and transformations?**
  - OpenMP to MPI: the **STEP** phase (ParMA European Project)
  - Heterogeneous code optimization for a hardware accelerator: **FREIA / SpoC** (ANR Project)
  - **GPU / CUDA**
  - **OpenCL** (FUI OpenGPU Project)
  - Generic vectorizer for SIMD instructions
  - Code generation for hardware accelerators (SCALOPES European Project)

STEP

V.1.1

- **STEP: Transformation System for Parallel Execution**
- **Use a single program to run both on shared-memory and distributed-memory architectures**
- **Parallelism specified via OpenMP directives**
- **A shared-memory OpenMP program is translated into a MPI program to run on distributed-memory machines**

OpenMP Directives

V.1.2

Parallel construct

Worksharing on loop i

```
#pragma omp parallel for shared(A, B, C)
private(i, j, k)
for (i = 0; i < N; i++) {
 for (j = 0; j < N; j++) {
  for (k = 0; k < N; k++) {
   C[i][j] = C[i][j] + A[i][k] * B[k][j];
  } } }
```

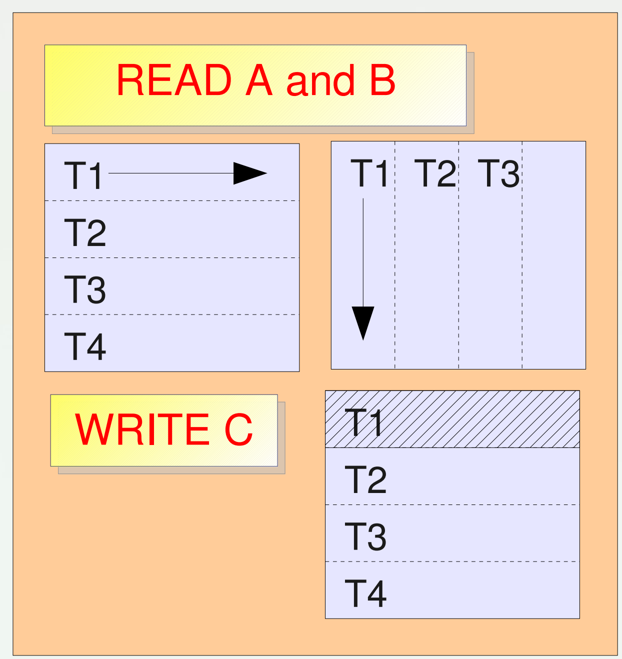
Using OpenMP:

- **The programmer** must guarantee that the code is correct
- … and avoid concurrent write access

Based on relaxed-consistency memory:

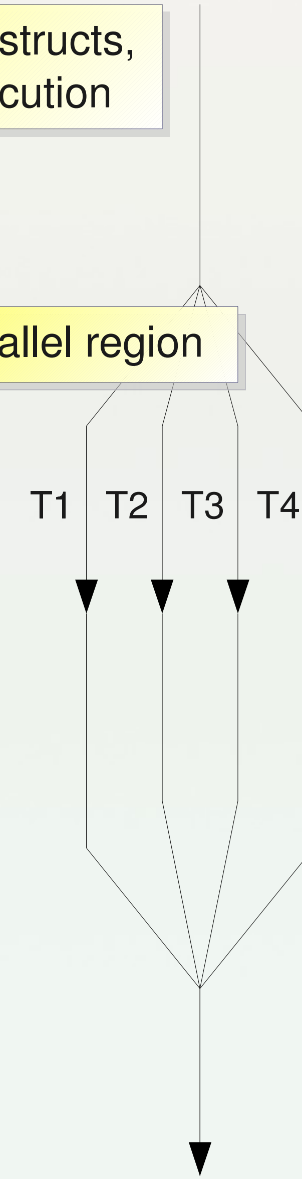
- **Update main memory** at specific points
- Explicit synchronisation primitives such as *flush*

Memory accesses in the worksharing region on loop i

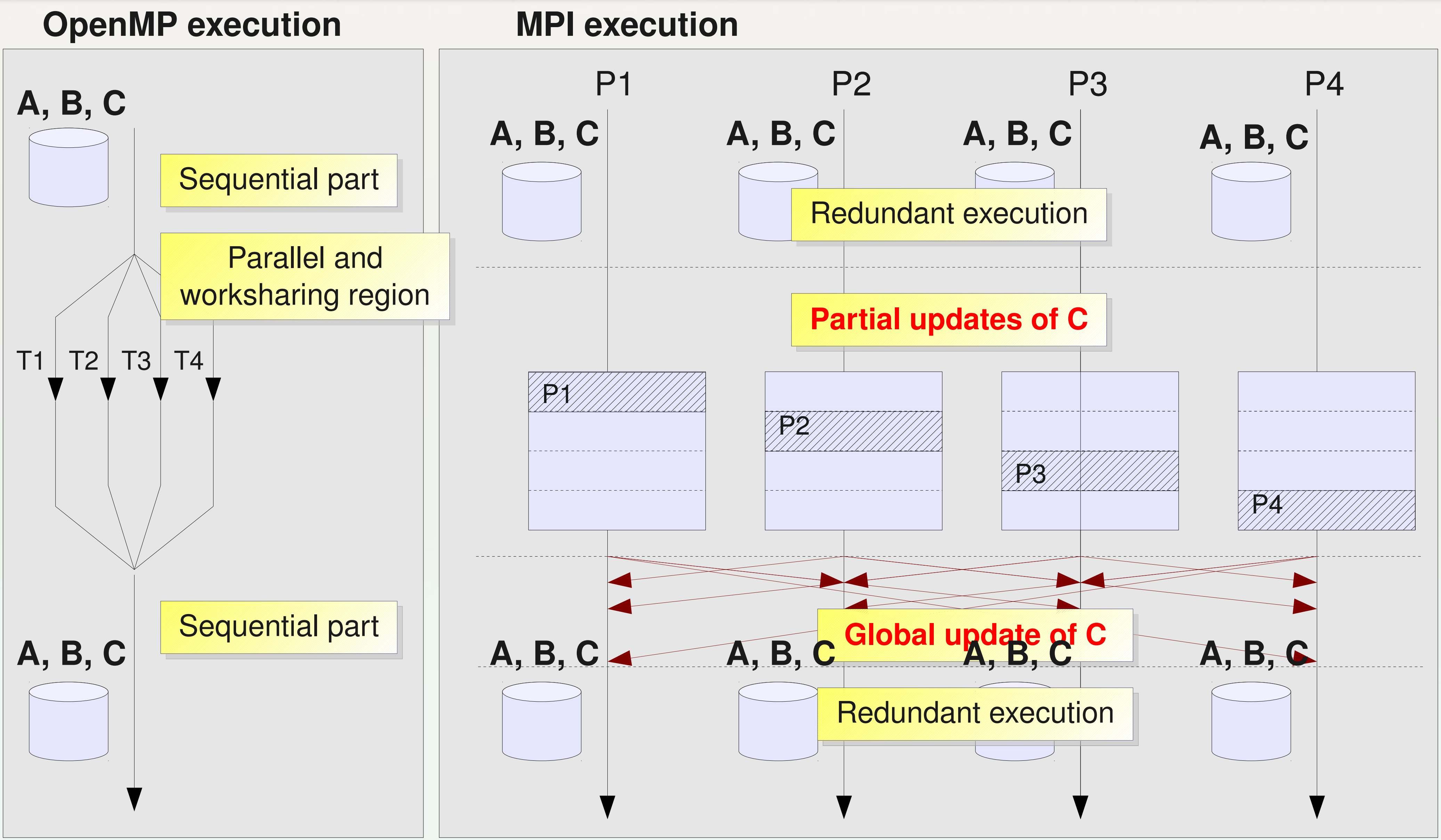


Outside parallel constructs, monothreaded execution

Parallel region



From a Shared-Memory to a Distributed-Memory Execution Model V.1.3



From OpenMP to MPI: three main phases

V.1.4

```
#pragma omp parallel for shared(A, B, C)
private(i, j, k)
for (i = 0; i < N; i++) {
 for (j = 0; j < N; j++) {
  for (k = 0; k < N; k++) {
    C[i][j] = C[i][j] + A[i][k] * B[k][j];
  } } }
```

1) Identify parallel constructs and compute worksharing

2) Global update: all2all communication

- Determine **modified data** inside the worksharing region for each process
- Find which process needs which data

3) Generate MPI code

SPMD message-passing programming

```
/*
  Explicit worksharing
  depending on process ID
*/

nbrows = N / nbprocs;
i_low = myrank * nbrows;
i_up =  (myrank + 1) * nbrows;

for (i = i_low; i < i_up; i++) {
  for (j = 0; j < N; j++) {
    for (k = 0; k < N; k++) {
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
  }
}

/* Explicit data update */
All2all_update(C);
```

Using PIPS for STEP

V.1.5

- **Interpreprocedural analyses**
  - Array regions as convex polyhedra
  - EXACT, MAY approximations
  - IN, OUT, READ, WRITE
- **PIPS as a workbench**
  - Intermediate representation
  - Program manipulation
  - Pretty-printer
  - Source-to-source transformation

Program Example

V.1.6

```
      PROGRAM MATMULT
      implicit none
      INTEGER  N, I, J, K
      PARAMETER (N=1000000)
      REAL*8 A(N,N), B(N,N), C(N,N)

      CALL INITIALIZE(A, B, C, N)

C       Compute matrix-matrix product
!$OMP PARALLEL DO
      DO 20 J=1, N
         DO 20 I=1, N
            DO 20 K=1, N
               C(I,J) = C(I,J) + A(I,K) * B(K,J)
 20         CONTINUE
!$OMP END PARALLEL DO

      CALL PRINT(C, N)
      END
```

Three PIPS modules:

- INITIALIZE
- PRINT
- MATMUL

One parallel loop in the MATMUL program

First PIPS phase: « STEP_DIRECTIVES »

V.1.7

- **Parse the OpenMP program:**
  - Recognize OpenMP directives
  - Outline OpenMP constructs in separate procedures

```
create myworkspace matmul.f

apply STEP_DIRECTIVES[%ALL]

close
```

"On all modules"

Output resources

```
step_directives    > PROGRAM.directives
                   > PROGRAM.outlined
                   > MODULE.code
                   > MODULE.callees
  ! MODULE.directive_parser
  < PROGRAM.entities
  < PROGRAM.outlined
  < PROGRAM.directives
  < MODULE.code
  < MODULE.callees
```

Input resources

« STEP_DIRECTIVES » results

V.1.8

MATMULT.f

```
      PROGRAM MATMULT
! MIL-STD-1753 Fortran extension not in PIPS
!     implicit none
      INTEGER  N, I, J, K
      PARAMETER (N=1000000)
      REAL*8 A(N,N), B(N,N), C(N,N)

     CALL INITIALIZE(A, B, C, N)
C     !$omp parallel do
     CALL MATMULT_PARDO20(J, 1, N, I, N, K, C, A, B)

     CALL PRINT(C, N)
     END
```

Initial MATMUL calling the outlined function

New module containing the parallel DO loop

MATMULT_PARDO20.f

```
      SUBROUTINE MATMULT_PARDO20(J, J_L, J_U, I, N, K, C, A, B)
      INTEGER J, J_L, J_U, I, N, K
      REAL*8 C(1:N, 1:N), A(1:N, 1:N), B(1:N, 1:N)
      DO 20 J = J_L, J_U
        DO 20 I = 1, N
          DO 20 K = 1, N
            C(I,J) = C(I,J)+A(I,K)*B(K,J)
20           CONTINUE
      END
```

Second PIPS Phase: « STEP_ANALYSE »

V.1.9

- **Parse the OpenMP program containing outlined functions**
- **For each outlined module corresponding to a OpenMP construct:**
  - Apply PIPS analyses: IN, OUT, READ, WRITE array regions
  - Compute SEND array regions describing data that have been modified by each process

```
create myworkspace matmul.f
activate MUST_REGIONS
activate TRANSFORMERS_INTER_FULL



apply STEP_DIRECTIVES[%ALL]
apply STEP_ANALYSE[%ALL]



close
```

MUST_REGIONS for the most precise analysis

TRANSFORMERS for accurate analysis
(translation of linear expressions...)

```
step_analyse                > PROGRAM.step_analyses
    < PROGRAM.entities
    < PROGRAM.directives
    < PROGRAM.step_analyses
    < MODULE.code
    < MODULE.summary_regions
    < MODULE.in_summary_regions
    < MODULE.out_summary_regions
```

Input/output resources

We ask for PIPS summary READ, WRITE, IN and OUT regions to be computed HERE!

« STEP_ANALYSE » Results

V.1.10

```
C  <C(PHI1,PHI2)-W-EXACT-{1<=PHI1, PHI1<=N, J_L<=PHI2, PHI2<=J_U}>
C  <C(PHI1,PHI2)-OUT-EXACT-{1<=PHI1, PHI1<=1000000, 1<=PHI2,
C    PHI2<=1000000, J_L==1, J_U==1000000, N==1000000}>

     SUBROUTINE MATMULT_PARDO20(J, J_L, J_U, I, N, K, C, A, B)
      INTEGER J, J_L, J_U, I, N, K
      REAL*8 C(1:N, 1:N), A(1:N, 1:N), B(1:N, 1:N)

      DO 20 J = J_L, J_U
         DO 20 I = 1, N
            DO 20 K = 1, N
               C(I,J) = C(I,J)+A(I,K)*B(K,J)
20             CONTINUE
      END
```

Print WRITE and OUT summary regions

WRITE regions

OUT regions

Compute SEND regions depending on loop bounds: WRITE ∩ OUT

```
C  <C(PHI1,PHI2)-write-EXACT-{1<=PHI1, PHI1<=N, PHI1<=1000000,
C    J_LOW<=PHI2, 1<=PHI2, PHI2<=J_UP, PHI2<=1000000}>
```

WRITE and OUT regions for array C
PHI1 (first dimension) is modified on all indices
PHI2 (second dimension) is modified between J_LOW and J_UP
SEND regions correspond to blocks of C rows

Third PIPS Phase: « STEP_COMPILE »

V.1.11

- **For each OpenMP directive**
  - Generate MPI code in outlined procedures (when necessary)

```
create myworkspace matmul.f
activate MUST_REGIONS
activate TRANSFORMERS_INTER_FULL



apply STEP_DIRECTIVES[%ALL]
apply STEP_ANALYSE[%ALL]
apply STEP_COMPILE[%MAIN]

close
```

```
step_compile     > PROGRAM.step_status
                 > MODULE.code
                 > MODULE.callees
   ! CALLEES.step_compile
   < PROGRAM.entities
   < PROGRAM.outlined
   < PROGRAM.directives
   < PROGRAM.step_analyses
   < PROGRAM.step_status
   < MODULE.code
```

Input/output resources

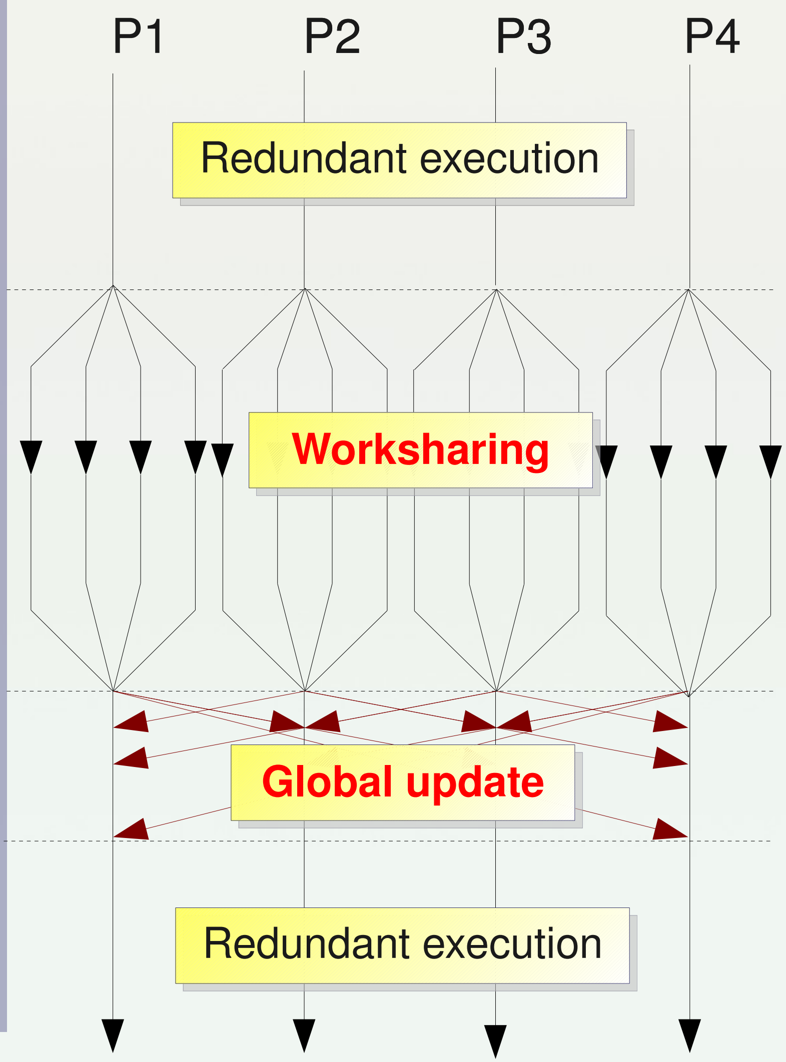
« STEP_COMPILE »: results

V.1.12

```
     SUBROUTINE MATMULT_PARDO20_HYBRID(J, J_L, J_U, I,
N, K, C, A, B)
C     Some declarations
     CALL STEP_GET_SIZE(STEP_LOCAL_COMM_SIZE_)
     CALL STEP_GET_RANK(STEP_LOCAL_COMM_RANK_)

     CALL STEP_COMPUTELOOPSLICES(J_LOW, J_UP, ...)
C     Compute SEND regions for array C
     STEP_SR_C(J_LOW,1,0) = 1
     STEP_SR_C(J_UP,1,0) = N
     ...
C     Where work is done...
     J_LOW = STEP_J_LOOPSLICES(J_LOW, RANK+1)
     J_UP = STEP_J_LOOPSLICES(J_UP,RANK_+1)
     CALL MATMULT_PARDO20_OMP(J, J_LOW, J_UP, I, N, K, C,
A, B)

!$omp master
     CALL STEP_ALLTOALLREGION(C, STEP_SR_C, ...)
!$omp end master
!$omp barrier
     END
```

3 different All2all: NONBLOCKING, BLOCKING1, BLOCKING2

Hybrid execution



Using STEP

V.1.13

Full *tpips* file

```
create myworkspace matmul.f
activate MUST_REGIONS
activate TRANSFORMERS_INTER_FULL
setproperty STEP_DEFAULT_TRANSFORMATION "HYBRID"
setproperty STEP_INSTALL_PATH " "

apply STEP_DIRECTIVES[%ALL]
apply STEP_ANALYSE[%ALL]
apply STEP_COMPILE[%MAIN]
apply STEP_INSTALL
close
```

"run_step.script" to run STEP on your source files

Get the transformed source in the Src directory

```
$ run_step.script matmul.f

$ ls matmul/matmul.database/Src
Makefile
matmul.f
MATMULT_PARDO20_HYBRID.f
MATMULT_PARDO20_OMP.f
MATMULT_PARDO20.f
STEP.h
steprt_f.h
step_rt/
```

Properties to tune STEP

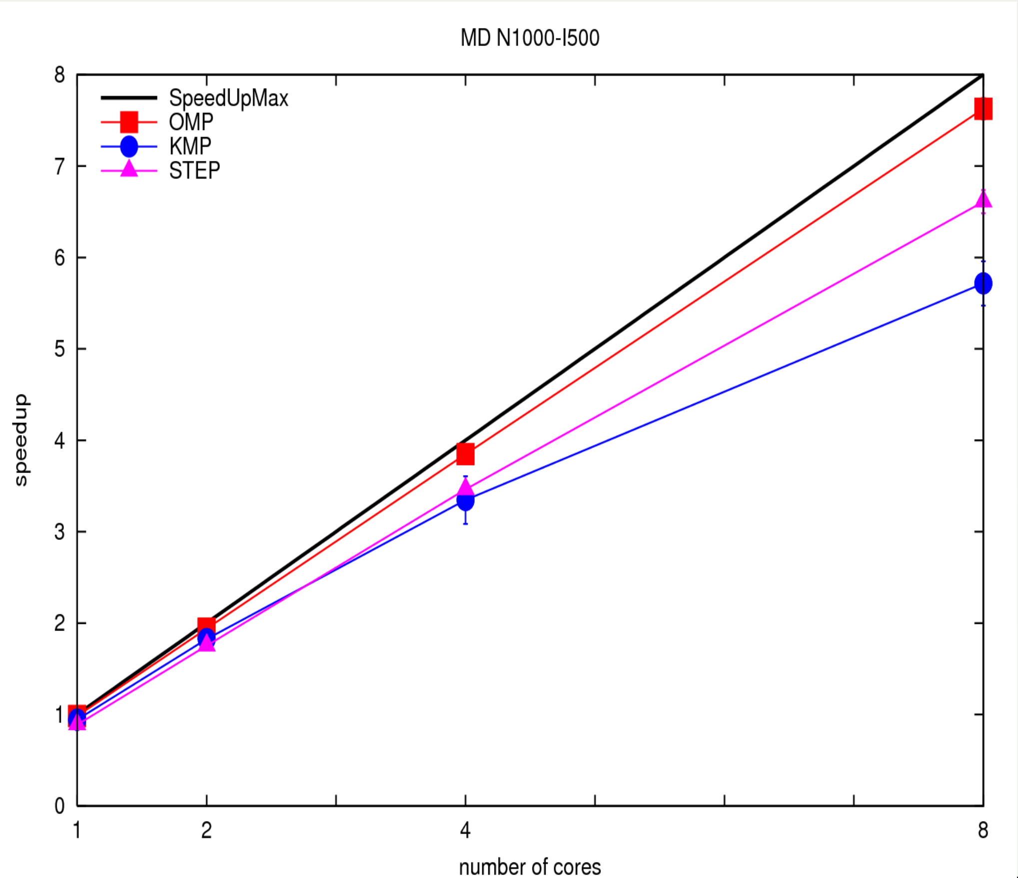
Different available transformations:

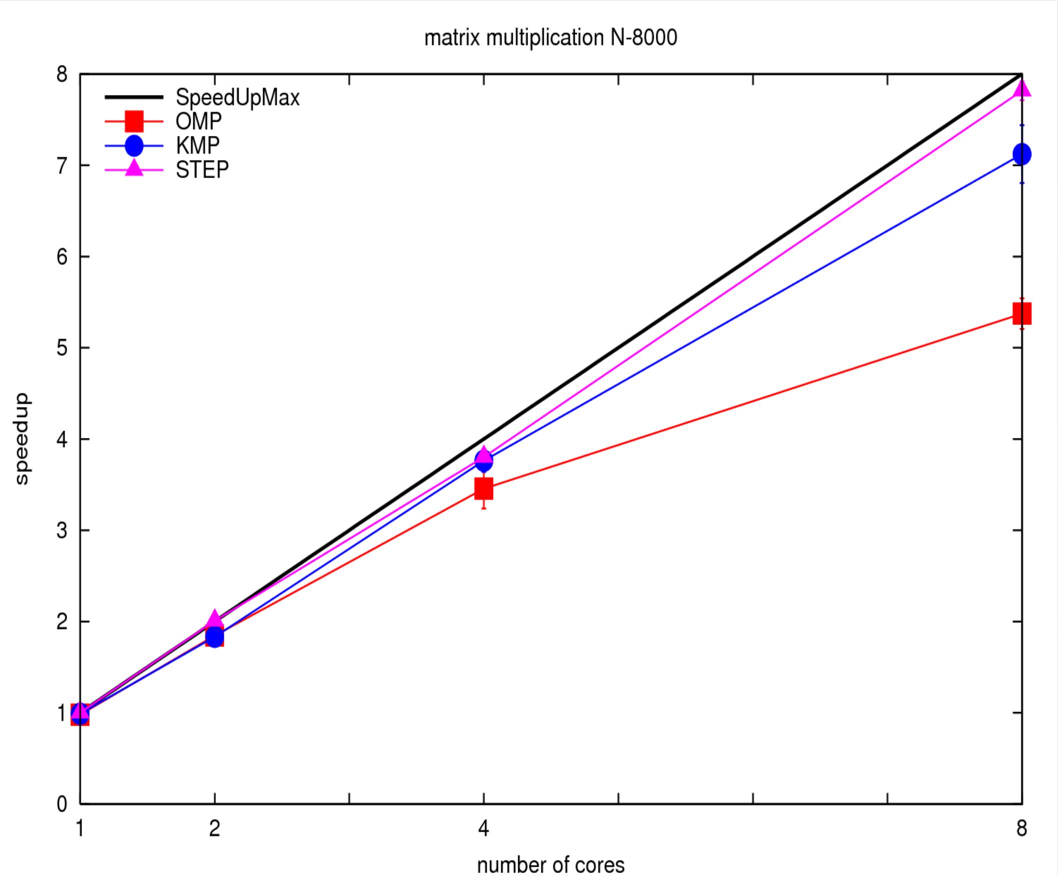
- MPI
- HYBRID
- OMP

Benchmarks: OpenMP / Intel Cluster OpenMP (KMP) / STEP

V.1.14

- **Transformations of some standard benchmarks:**
  - Transformation is correct and run in every case
  - Good performance for coarse-grain parallelism
  - Poor performance when parallel constructs are inserted inside loops





STEP: Conclusion and Perspectives

V.1.15

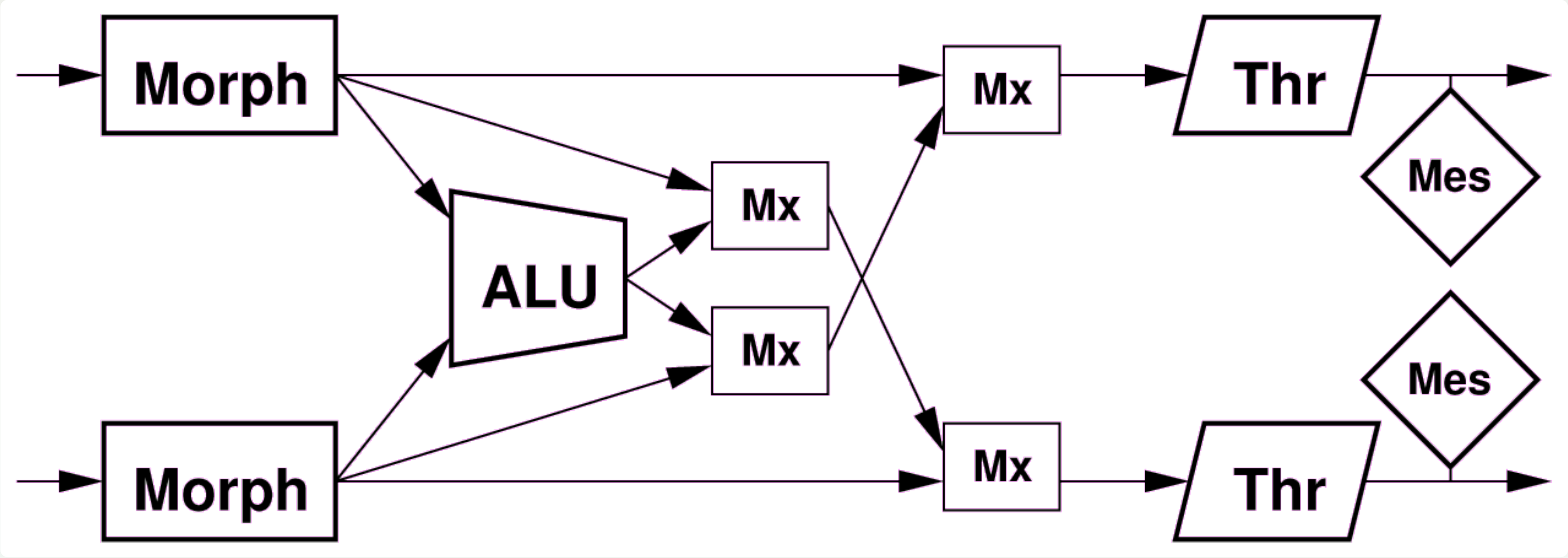
- **The automatic transformation from OpenMP to MPI is efficient in several cases**
- **... thanks to PIPS interprocedural array regions analyses**

- **Future work**
  - Exploit more results of PIPS analyses to improve performance when parallel loops are nested inside sequential loops
  - Some OpenMP features are not handled yet, such as “critical” or “schedule”

SPoC: FREIA ANR Project 2008-2010

V.2.1

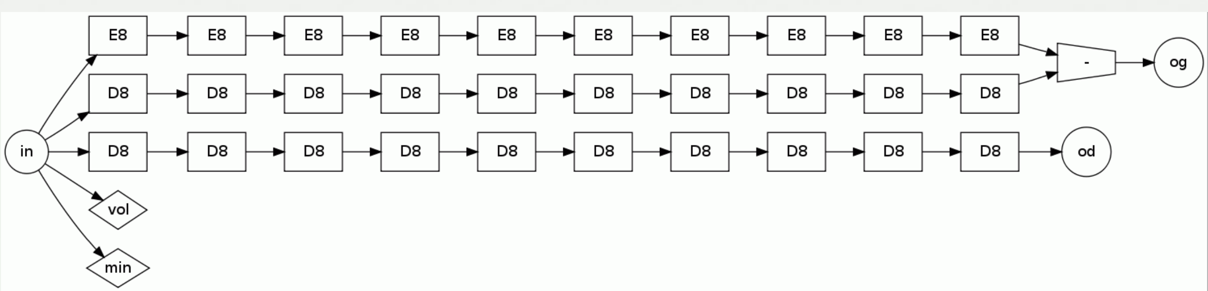
- **Input: Image Library API**
- **Targets: FPGA-based hardware accelerators**
  - SPoC image-vector processor (8 pipelined stages)
  - Terapix SIMD processor (128 PE)
  - ...

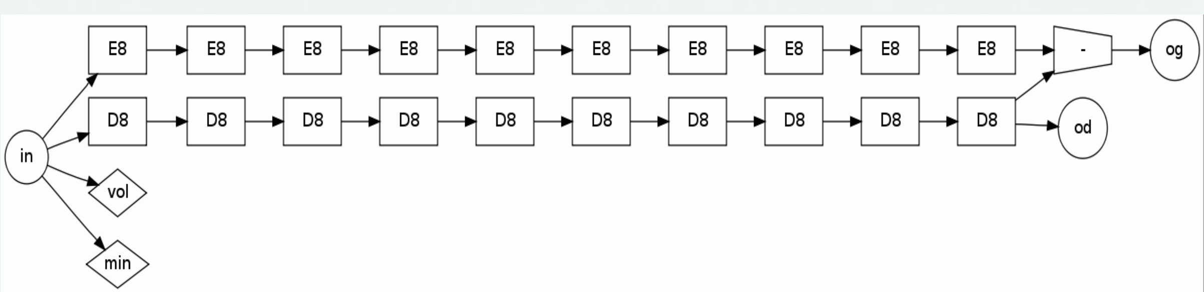


FREIA SPoC Compiler

V.2.2

1. **Build image expression DAG**
   - Inlining, constant propagation, dead code elimination, flattening and loop unrolling
2. **Optimize DAG**
   - CSE, intermediate images… (~4500 LoC)
3. **Generate SPoC configuration**





High-level API: 4 calls

```
freia_global_min(in, &min);
freia_global_vol(in, &vol);
freia_dilate(od, in, 8, 10);
freia_gradient(og, in, 8, 10);
```

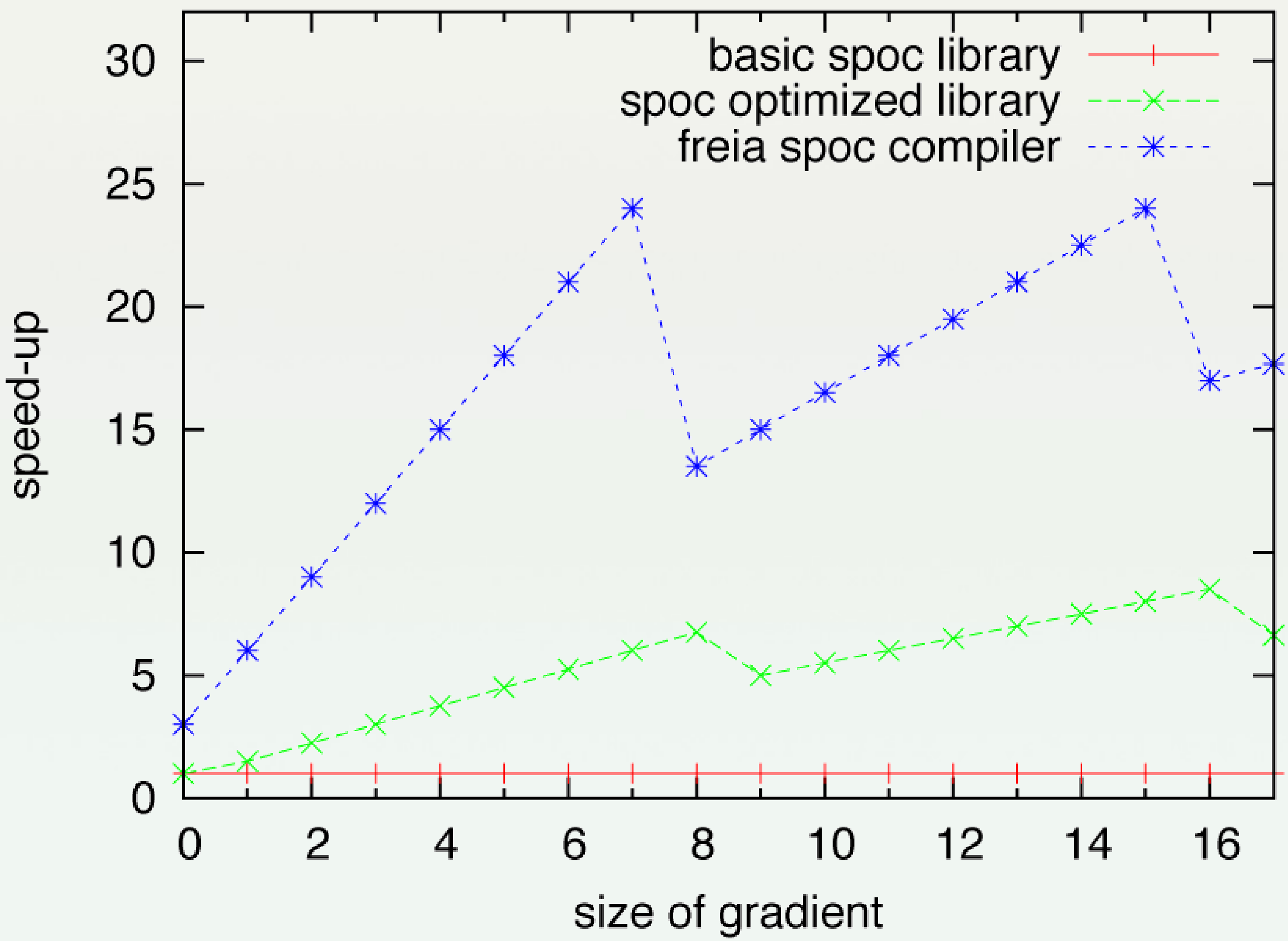
Basic API: 33 calls

```
freia_aipo_global_min(in, &min);
freia_aipo_global_vol(in, &vol);
freia_aipo_dilate_8c(od, in, k8c);
freia_aipo_dilate_8c(od, od, k8c);
...
freia_aipo_dilate_8c(tmp, in, k8c);
freia_aipo_dilate_8c(tmp, tmp, k8c);
...
freia_aipo_erode_8c(og, in, k8c);
freia_aipo_erode_8c(og, og, k8c);
...
freia_aipo_sub(og, tmp, og);
```

FREIA SPoC Compiler Results

V.2.3

- **Up to 24x speedup!**



Par4All for CUDA

V.3.1

Par4All

Par4All

V.3.2

- **HPC Project is a company that produces Wild Systems desk-side accelerators with manycore, GPU or FPGA**
  - Needs tools to port customer applications on its Wild Systems boxes
  - HPC Project also offers training in parallel computing and professional services in application optimization & parallelization
- **Developing tools is quite a difficult business...**
- **→ Launch of a platform to merge various open source developments : Par4All**
  - Migration of sequential software to multicore and other parallel processors without using any specific programming language
  - HPC Project committed to develop & promote PIPS
    - Integration process of the platform's software components
    - Federating contributions & release production

  → New ways for the software industry to address more easily intensive computation domains where parallelism is a determining factor

http://par4all.org http://wild-systems.com http://hpc-project.com

Par4All for CUDA

- **GPU computing: quite power-efficient alternative**
  - Over 1 TFLOPS per card (nVidia GT 200, AMD Radeon 5870...)
  - Good memory bandwidth (150 GB/s)
- **Programming tools are improving (CUDA, OpenCL...) but still difficult to get started with**
- **→ Need tools to increase programmer productivity!**
- **Using PIPS infrastructure to automate CUDA code generation**
  - CUDA from C...
  - ... & CUDA from Fortran!

Basic GPU Execution Model

- A sequential program on a host launches computational-intensive kernels on a GPU
- Allocate storage on the GPU
- Copy-in data from the host to the GPU
- Launch the kernel on the GPU
- The host waits...
- Copy-out the results from the GPU to the host
- Deallocate the storage on the GPU

Challenges in Automatic CUDA Generation

V.3.5

- **Find parallel kernels**
- **Improve data reuse inside kernels**
  - to have better computational intensity
  - even if the memory bandwidth is quite higher than on a CPU…
- **Address the memory in a GPU-friendly way, to coalesce accesses**
- **Take advantage of complex memory hierarchy**
  - shared memory, cached texture memory, registers…
- **Reduce the copy-in and copy-out transfers on the PCIe**
- **Reduce memory usage in the GPU**
  - no swap there, yet…
- **Reduce inter-block synchronizations**
- **Overlap computations and GPU-CPU transfers (via streams)**

Parallelization

- **Find parallelism using one of the various PIPS parallelization phases**
- **Use coarse-grain parallelization based on array regions used by different loop iterations because:**
  - It generates GPU-friendly coarse-grain parallelism
  - It accept complex control code without if-conversion

Outlining

V.3.7

- **Parallel code with kernel code on GPU**
  - Need to extract parallel source code into kernel source code: outlining of parallel loop-nests

Before:

```
for(i = 1; i <= 499; i ++)
    for(j = 1; j <= 499; j ++) {
        save[i][j]=0.25*(space[i-][j]+space[i+][j]+space[i][j-1]+space[i][j+1])
    }
```

After:

```
p4a_kernel_launcher_0(space,save);
[...]
void p4a_kernel_launcher_0(float_t space[SIZE][SIZE], float_t save[SIZE][SIZE]) {
    for(i=1;i<=499;i+=1)
        for(j=1;j<=499;j+=1)
            p4a_kernel_0(i,j,save,space);
}
[...]
void p4a_kernel_0(float_t space[SIZE][SIZE], float_t save[SIZE][SIZE], int i, int j) {
    save[i][j]=0.25*(space[i-1][j]+space[i+1][j]+space[i][j-1]+space[i][j+1]);
}
```

From Array Regions to GPU Memory Allocation

V.3.8

- **Memory accesses are summed up for each statement as regions**
  - Integer polytope lattice
- **There are regions for write accesses and regions for read accesses**
- **The regions can be**
  - exact if PIPS can prove that only these points are accessed,
  - or inexact if PIPS can only find an over-approximation of what is really accessed

```
for(i=0;i<=n−1;i+=1)
    for(j=i;j<=n−1;j+=1)
        h_A[i][j]=1;
```

```
// <h_A[PHI1] [PHI2]-W-EXACT-{0<=PHI1, PHI2+1<=n, PHI1<=PHI2}>
    for(i=0;i<=n−1;i+=1)
// <h_A[PHI1] [PHI2]-W-EXACT-{PHI1==i , i<=PHI2, PHI2+1<=n, 0 <=i}>
        for(j=i;j<=n−1;j+=1)
// <h_A[PHI1] [PHI2]-W- EXACT-{PHI1==i , PHI2==j , 0<=i , i<=j , 1+j<=n}>
            h_A[i][j]=1;
```

- **These read/write regions for a kernel are used**
  - to allocate with a cudaMalloc() in the host code the memory used inside a kernel
  - and to deallocate it later with a cudaFree()

Communication Generation

V.3.9

- **Conservative approach to generate communications:**
  - Associate any GPU memory allocation with a copy-in to keep its value in sync with the host code
  - Associate any GPU memory deallocation with a copy-out to keep the host code in sync with the updated values on the GPU
- **But a kernel could use an array as a local (private) array**
  - ...PIPS does have many privatization phases
- **But a kernel could initialize an array, or use the initial values without writing into it or use/touch only a part of it or...**
- **PIPS gives two very interesting region types for this purpose**
  - In-region abstracts what is really needed by a statement
  - Out-region abstracts what is really produced by a statement to be used later elsewhere
- **In-Out regions can directly be translated with CUDA into**
  - copy-in: cudaMemcpy(accel_address,host_address, size, cudaMemcpyHostToDevice)
  - copy-out: cudaMemcpy(host_address,accel_address, size, cudaMemcpyDeviceToHost)

Loop Normalization

V.3.10

- **Hardware accelerators use fixed iteration space:**
  - CUDA thread index starting from 0…
- **Parallel loops: more general iteration space**
- **Loop normalization**
- **Before:**

```
for(i=1;i<SIZE-1;i++)
  for(j=1;j<SIZE-1;j++)
    save[i][j]=0.25*(space[i-1][j]+space[i+1][j]+space[i][j-1]+space[i][j+1]);
```

- **After:**

```
for(i=0;i<SIZE-2;i++)
  for(j=0;j<SIZE-2;j++)
    save[i+1][j+1]=0.25*(space[i][j+1]+space[i+2][j+1]+space[i+1][j]+space[i+1][j+2])
```

Par4All Accel Runtime - The Big Picture

V.3.11

- **CUDA cannot be directly represented in PIPS IR**
  - e.g. __device__ or <<< >>>
- **PIPS motto: "keep the IR as simple as possible"**
- **Use calls to intrinsics functions that can be represented directly**
- **Intrinsics functions are implemented with (macro-)functions**
- **p4a_accel.h currently has two implementations:**
  - p4a_accel-CUDA.h can be compiled with CUDA for nVidia GPU execution or emulation on CPU
  - p4a_accel-OpenMP.h can be compiled with an OpenMP compiler for simulation on a (multicore) CPU

Par4All Accel Runtime - The Big Picture

V.3.12

```
int main(int argc, char *argv[]) {
[...]
  float_t (*p4a_var_space)[SIZE][SIZE];
  P4A_accel_malloc(&p4a_var_space, sizeof(space));
  P4A_copy_to_accel(space, p4a_var_space, sizeof(space));

  float_t (*p4a_var_save)[SIZE][SIZE];
  P4A_accel_malloc(&p4a_var_save, sizeof(save));
  P4A_copy_to_accel(save, p4a_var_save, sizeof(save));

  for(t = 0; t < T; t++)
    compute(*p4a_var_space, *p4a_var_save);

  P4A_copy_from_accel(space, p4a_var_space, sizeof(space));

  P4A_accel_free(p4a_var_space);
  P4A_accel_free(p4a_var_save);
[...]
}
```

```
void compute(float_t space[SIZE][SIZE],
             float_t save[SIZE][SIZE]) {
[...]
  p4a_kernel_launcher_0(space, save);
[...]
}
```

Par4All Accel Runtime - The Big Picture

V.3.13

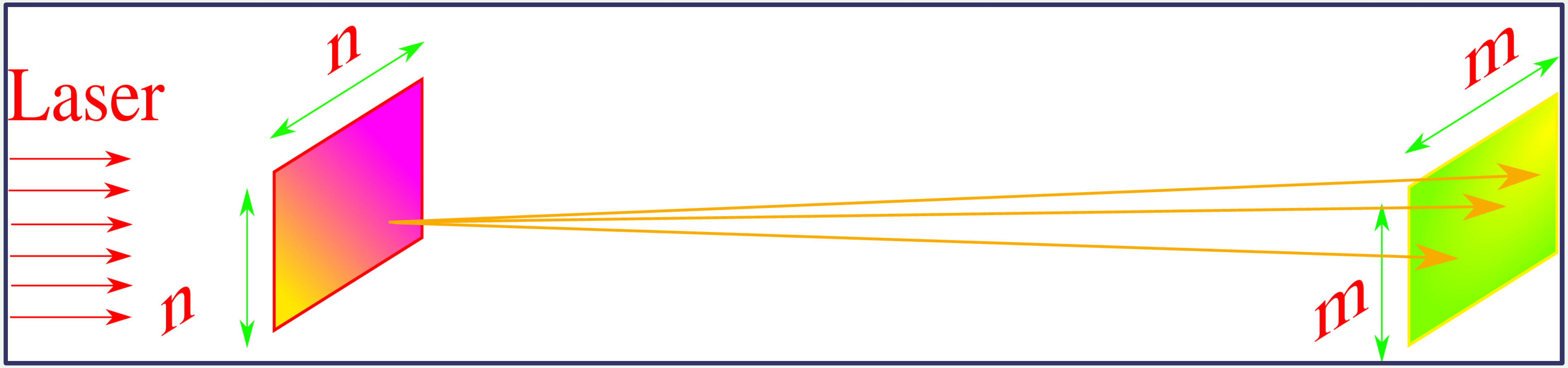
```
void p4a_kernel_launcher_0(float_t space[SIZE][SIZE], float_t save[SIZE][SIZE]) {
  P4A_call_accel_kernel_2d(p4a_kernel_wrapper_0, SIZE, SIZE, space, save);
}
P4A_accel_kernel_wrapper void
p4a_kernel_wrapper_0(float_t space[SIZE][SIZE],
                     float_t save[SIZE][SIZE]) {
  int j, i;
  i = P4A_vp_0;
  j = P4A_vp_1;
  if (i >= 1 && i <= SIZE - 1 && j >= 1 && j <= SIZE - 1)
    p4a_kernel_0(space, save, i, j);
}
```

```
P4A_accel_kernel void p4a_kernel_0(float_t space[SIZE][SIZE],
                                   float_t save[SIZE][SIZE],
                                   int i,
                                   int j) {
  save[i][j] = 0.25*(space[i-1][j]+space[i+1][j]
                     +space[i][j-1]+space[i][j+1]);
}
```

Holotetrix' Hologram Verification

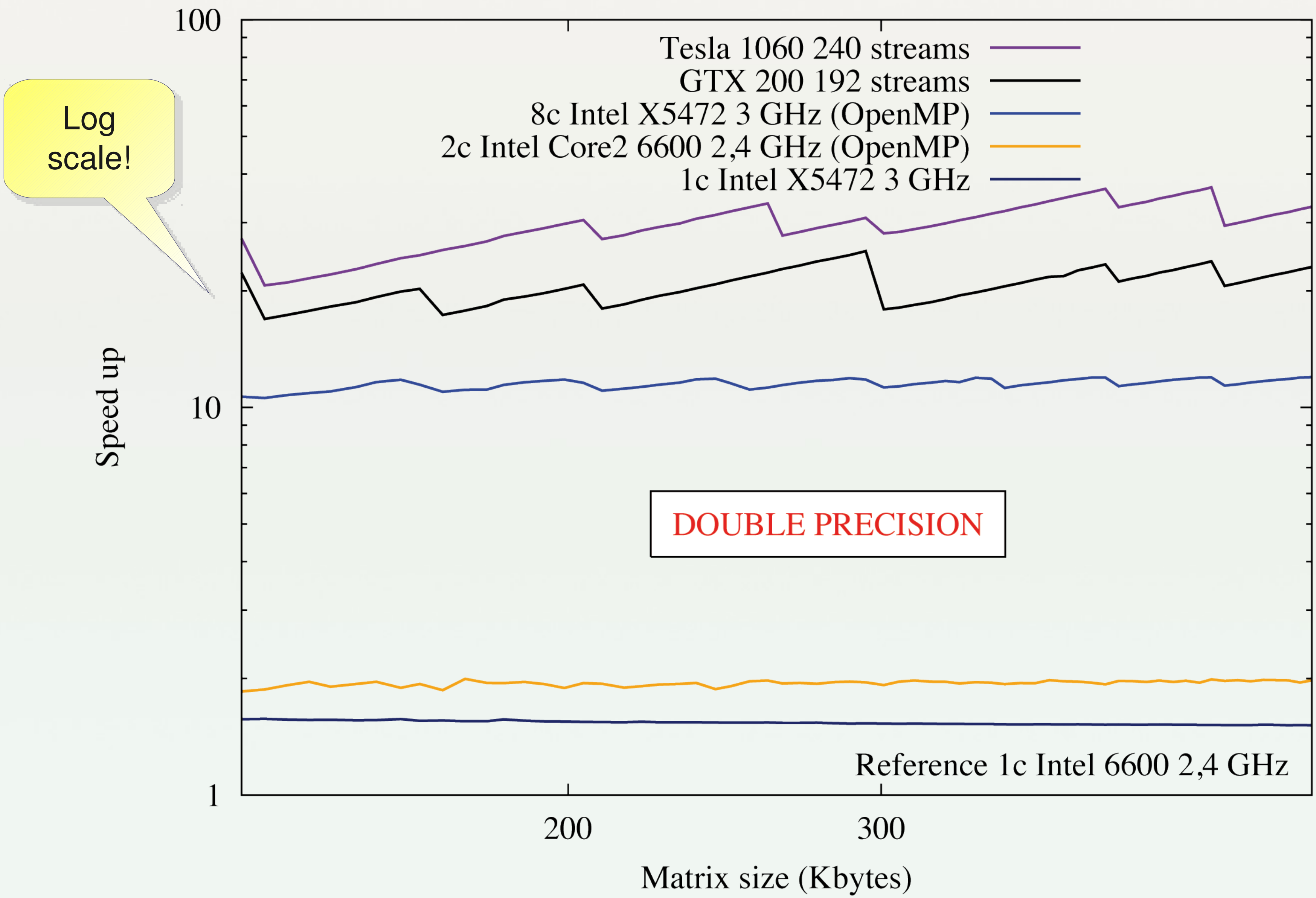
V.3.14

- **Holotetrix's primary activities: http://www.holotetrix.com**
  - Design, fabrication and commercialization of prototype diffractive optical elements (DOE) and micro-optics
  - Diverse industrial applications such as LED illumination, laser beam shaping, wavefront analyzers, etc.
  - Hologram verification with direct Fresnel simulation
- **Program in C parallelized with**
  - Par4All CUDA and CUDA 2.3, Linux Ubuntu x86-64
  - Par4All OpenMP, gcc 4.3, Linux Ubuntu x86-64
- **Reference: Intel Core2 6600 @ 2.40GHz**



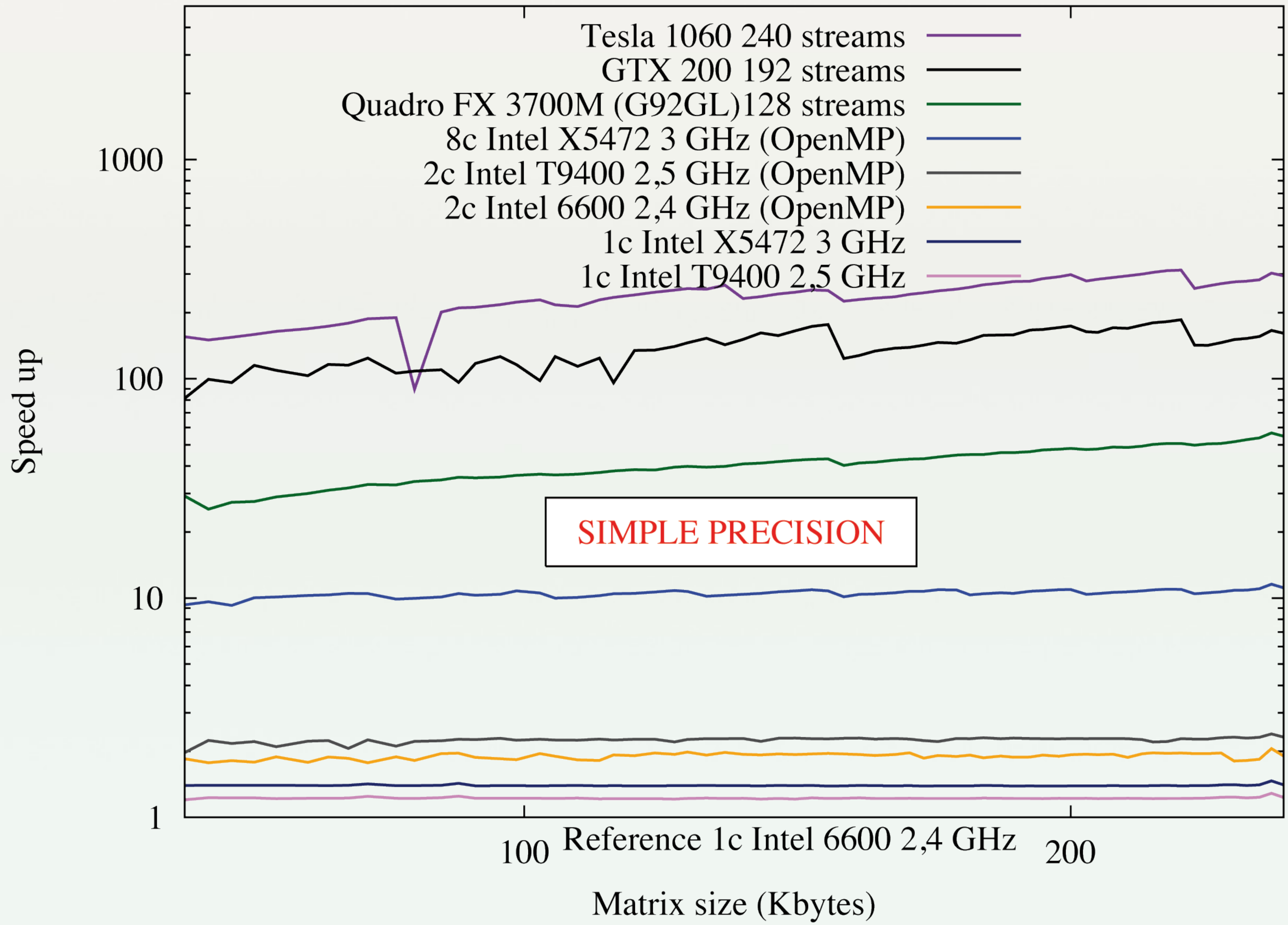
Performance Results with Holotetrix' Application

V.3.15



Performance Results with Holotetrix' Application

V.3.16



VI. Conclusion

VI. Conclusion

VI. Conclusion

VI.0.2

- **Many analyses and transformations**
- **Ready to be combined for new projects**
- **Interprocedural source-to-source tool**
- **Automatic consistency and persistence management**
- **Easy to extend: a matter of hours, not days!**
- **PIPS is used, developped and supported by different institutions:**
  - MINES ParisTech, TELECOM Bretagne, TELECOM SudParis, HPC Project, ...
- **Used in several on-going projects:**
  - FREIA, OpenGPU, SCALOPES, Par4All...
- **May seem difficult to dominate**
  - A little bit of effort at the beginning saves a lot of time

PIPS Future Work

- **Full support of C:**
  - Semantics analyses extended to structures and pointers
  - Points-to analysis
  - Convex array regions extended to struct and pointers
- **Support of Fortran 95 (using gfortran parser)**
- **Code generators for specific hardware:**
  - CUDA
  - OpenCL
  - SSE
  - Support for FPGA-based hardware accelerator
  - Backend for a SIMD parallel processor
- **Optimization of the OpenMP to MPI translation**

PIPS Online Resources

VI.0.4

- **Website:** http://pips4u.org
  - **Documentation:**
    - **Getting Started** (examples from the Tutorial)
    - **Guides and Manuals** (PDF, HTML):
      - Developers Guide
      - Tpips User Manual
      - Internal Representation for Fortran and C
      - PIPS High-Level Software Interface • Pipsmake Configuration
- **SVN repository:** http://svn.pips4u.org/svn
- **Debian packages**: http://ridee.enstb.org/debian/
- **Trac site:** http://svn.pips4u.org/trac
- **IRC:** irc://irc.freenode.net/pips
- **Mailing lists:** pipsdev at cri.mines-paristech.fr (developer discussions)
  pips-support at cri.mines-paristech.fr (user support)

Credits

- **Laurent Daverio**
  - Coordination and integration
  - Python scripts for OpenOffice slide generation
- **Corinne Ancourt**
- **Fabien Coelho**
- **Stéphanie Even**
- **Serge Guelton**
- **François Irigoin**
- **Pierre Jouvelot**
- **Ronan Keryell**
- **Frédérique Silber-Chaussumier**
- **And all the PIPS contributors...**

Python scripts for Impress (Open Office)

- **Include files**
- **Colorize files**
- **Compute document outline**
- **Visualize the document structure**