La bibliothèque **Nestor**
Un outil pour l'implementation de transformations source à source de codes Fortran

Georges-André SILBER
Alain DARTE

Introduction and motivations

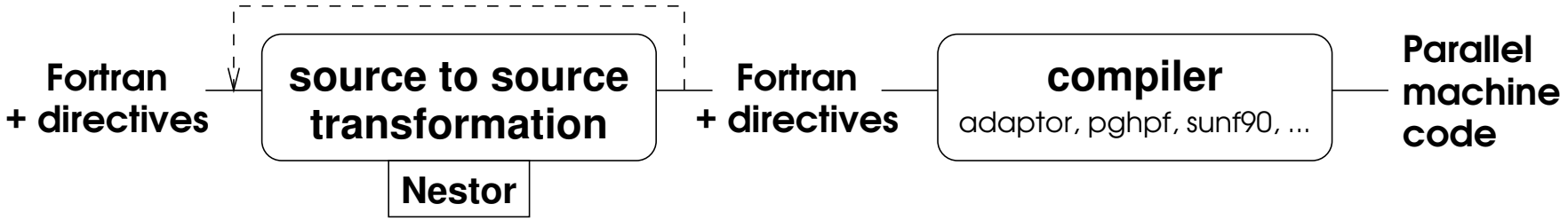
Parallelizing algorithms $\neq$ Parallelizing compilers

- Implementation is difficult and time consuming.
- Implementation is often a research problem!
- Technical aspects of the implementation must be eased.

$\Rightarrow$ Simple source code representation, easy manipulation.

Introduction and motivations (cont'd)

- Source to source transformations $\rightarrow$ DIRECTIVES.



- Fortran (77,90) + (HPF 2.0, OpenMP, CrayMP, SunMP)
- Source code is still human-readable.
- Output can become input (filters).

Nestor

- Nestor is a C++ library providing classes and methods. Features:
  - parse a Fortran source code + directives
  - give a high level representation of the AST
  - transform the AST easily
  - compute dependences
  - unparse a Fortran source code + directives

Related tools

- Toy implementations (Bouclettes, Petit, LooPo).
- Huge systems (PIPS, Polaris, SUIF, PAF).
- Nestor does nothing! But:
  - it is SMALL (basic blocks),
  - it is FULLY DOCUMENTED (online with HTML and Java),
  - it offers a Fortran input and output (+ its dialects).

▶ High level language + directives, high level representation ◀

Implementation choices

- C++
  - widely used (external libraries)
  - object-oriented (inheritance, specific methods and attributes)
  - templates, lists, types (STL)
  - overloading («)
  - virtual methods

Transformation pass

```
#include <libnstbase.H>

void main(int argc, char** argv)
{
  if (argc == 2)
    {
      NstComputationUnit prog(argv[1]);
      cout << "! file:" << argv[1] << endl;
      cout << prog << endl;
    }
}
```

Internal representation

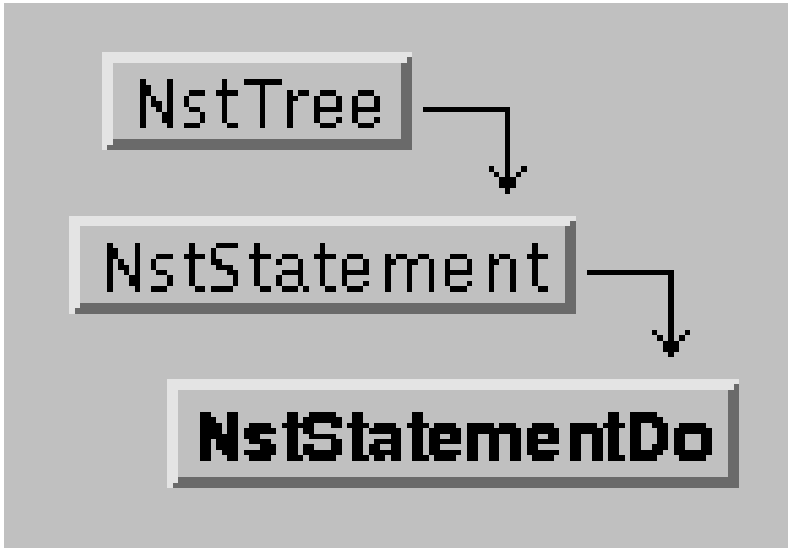
```
class NstStatementDo : public NstStatement {
public:
  NstVariable* index() const;
  NstVariable* index(NstVariable& new_var);
  NstExpression* lower_bound() const;
  NstExpression* lower_bound(NstExpression& new_exp);
  NstExpression* upper_bound() const;
  NstExpression* upper_bound(NstExpression& new_exp);
  NstStatementList body;
  bool independent;
  NstVariableList new_variables;
  ...
};
```

Type of objects

- Each class inherit from `NstTree` and has a `type()`.

`NstStatementContinue`	`NstStatementBasic`	`NstStatementWhere`
`NstStatementWhile`	`NstStatementIf`	`NstStatementNestor`
`NstStatementDo`	`NstStatementForall`	

- Example: inheritance graph of `NstStatement`.



Type of objects (cont'd)

- Example: prints the number of `INDEPENDENT` loops (no recursion).

```
void print_number_of_par(NstStatementList& sl)
{
  int num = 0;
  NstStatementList::iterator sl_it;
  NstStatement* current_stat;
  NstStatementDo* do_loop;
  for (sl_it = sl.begin(); sl_it != sl.end(); sl_it++) {
    current_stat = *sl_it;
    if (current_stat->type() == NST_STATEMENT_DO) {
        do_loop = (NstStatementDo*) current_stat;
        if (do_loop->independent) ++num;
    }
  }
  cout << "There are " << num << " parallel loops." << endl;
}
```

Expressions

- Operators are overloaded (`a = b + c`).

```
friend NstExpressionBinary& operator +  (NstExpression& l, NstExpression& r);
friend NstExpressionBinary& operator -  (NstExpression& l, NstExpression& r);
friend NstExpressionBinary& operator *  (NstExpression& l, NstExpression& r);
friend NstExpressionBinary& operator /  (NstExpression& l, NstExpression& r);
friend NstExpressionBinary& operator ^  (NstExpression& l, NstExpression& r);
friend NstExpressionBinary& operator <  (NstExpression& l, NstExpression& r);
friend NstExpressionBinary& operator >  (NstExpression& l, NstExpression& r);
friend NstExpressionBinary& operator >= (NstExpression& l, NstExpression& r);
friend NstExpressionBinary& operator <= (NstExpression& l, NstExpression& r);
friend NstExpressionBinary& operator && (NstExpression& l, NstExpression& r);
friend NstExpressionBinary& operator || (NstExpression& l, NstExpression& r);
friend NstExpressionBinary& operator == (NstExpression& l, NstExpression& r);
friend NstExpressionBinary& operator != (NstExpression& l, NstExpression& r);
```

Traversals

- Virtual methods: `init()` and `next()`.

```
void print_type(NstTree* t)
{
  cout << t->class_type() << endl;
  NstTree* current = t->init();
  while (current) {
    print_type(current);
    current = t->next();
  }
}
```

- Virtual method `traversal()`.

```
void traversal(int (*action)(NstTree*, void*), void* extra = NULL);
```

Traversals (cont'd)

- Class `NstTreeTraversal`

```
class NstTreeTraversal {
public:
  virtual int pre_action(NstTree* t);
  virtual int in_action(NstTree* t, NstTree* st, int num);
  virtual void post_action(NstTree* t);
  virtual void traversal(NstTree* t);
};
```

Traversals (cont'd)

- `void traversal(NstTree* t);`

```
void NstTreeTraversal::traversal(NstTree* t) {
  if (pre_action(t) == 0) {
    int stn = 0;
    NstTree* current = t->init();
    while (current) {
      stn++;
      if (in_action(t, current, stn) == 0)
        traversal(current);
      current = t->next();
    }
  }
  post_action(t);
}
```

On-line documentation (cont'd)

- **virtual void traversal(NstTree* t)**

  Recursive traversal of the tree. Here is the source code of the traversal method:

```
void NstTreeTraversal::traversal(NstTree* t)
{
  if (pre_action(t) == 0)
    {
      int stn = 0;
      NstTree* current = t->init();
      while (current)
        {
          stn++;
          if (in_action(t, current, stn) == 0)
            traversal(current);
          current = t->next();
        }
    }
  post_action(t);
}
```

Front-end

- Supports Fortran 77 and Fortran 90 with HPF 2.0 directives.
- Modified version of Adaptor's front-end.
- New directives (semi-automatic parallelization).
- Parsing a code:

```
NstComputationUnit program("toto.hpf", options);
```

Back-end

- Supports Fortran (HPF 2.0, OpenMP, CrayMP and SunMP), and Petit.
- Each class has its own unparse method («).
- Example: unparse method of `NstStatementIf`.

```
void NstStatementIf::unparse_fortran(ostream& s) const
{
  indent(s); s << "if (" << *condition() << ") then" << endl;
  inc_indent(); s << then_part << endl; dec_indent();
  if (else_part.size() > 0) {
    indent(s); s << "else" << endl;
    inc_indent();
    s << else_part << endl;
    dec_indent();
  }
  indent(s); s << "end if";
}
```

Dependences and graphs

- Nestor uses Petit and the Polylib (level, direction, polyhedral).
- Any part of the code can be passed to Petit by file (`NstRDGVar`).
- Example:

```
   DO I=1,n
      DO J=1,n
S:       A(i,j) = A(j,i) + A(i,j-1)
      ENDDO
   ENDDO
```

Examples: printing unit names

```
# include <libnstbase.H>

int print_unit_name(NstTree* t, void* ignored) {
  NstUnit* unit = t->isNstUnit();
  if (unit) cout << *unit->name() << endl;
  return 0;
}

void main(int argc, char** argv)
{
  if (argc == 2) {
      NstComputationUnit file(argv[1]);
      file.traversal(print_unit_name);
  }
}
```

Examples: renaming a variable

```
void rename_all_identifiers(NstUnitSubroutine& us)
{
  NstSymbolTable sb = us.object()->symbols();
  NstTree* current;
  NstIdentifier* name;

  current = sb.init();
  while (current) {
    if (current->type() == NST_OBJECT_VARIABLE) {
      name = new NstIdentifier();
      ((NstObject*)current)->identifier(*name);
    }
    current = sb.next();
  }
}
```

Examples: checking parallelism

```
int check_level(NstRDGVar& dg, int level)
{
  NstEdgeList::iterator el_it;
  NstDependence* dep;

  for (el_it = dg.edges.begin(); el_it != dg.edges.end(); el_it++) {
    dep = (NstDependence*) (*el_it);
    if (dep->level == level) return 1;
  }
  return 0;
}
```

Examples: checking parallelism (cont'd)

```
int parallel_loop(NstTree* t, void* ignored)
{
  NstStatementDo* stdo = NULL;
  if (t->type() == NST_STATEMENT_DO) stdo = (NstStatementDo*) t;
  if (stdo) {
    NstRDGVar dep_graph(*stdo);
    if (dep_graph.built()) { // graph successfully built
      stdo->independent = !check_level(dep_graph, 1);
    }
  }
  return 0;
}
```

Examples: checking parallelism (cont'd)

```
void main(int argc, char** argv)
{
  if (argc == 2) {
    NstComputationUnit file(argv[1]);
    file.traversal(parallel_loop);
    cout << file << endl;
  }
}
```

Allen and Kennedy with control dependences

```
do I = 1,N
  do J = 2,N
    do K = 2, A(I,J+1,1)
      A(I,J,K) = B(I,J,K)+A(I,J-1,K)
      C(I,J,K) = A(I,J-1,K-1)+D(I,J,K)
    enddo
  enddo
enddo
```

Concluding remarks

- Nestor is available:

  `http://www.ens-lyon.fr/~gsilber/nestor`

- We are implementing parallelization algorithms.