Scaling up Partial Evaluation for Optimizing The Sun Commercial RPC Protocol

Gilles Muller Nic Volanschi Renaud Marlet
COMPOSE group
IRISA / INRIA - University of Rennes

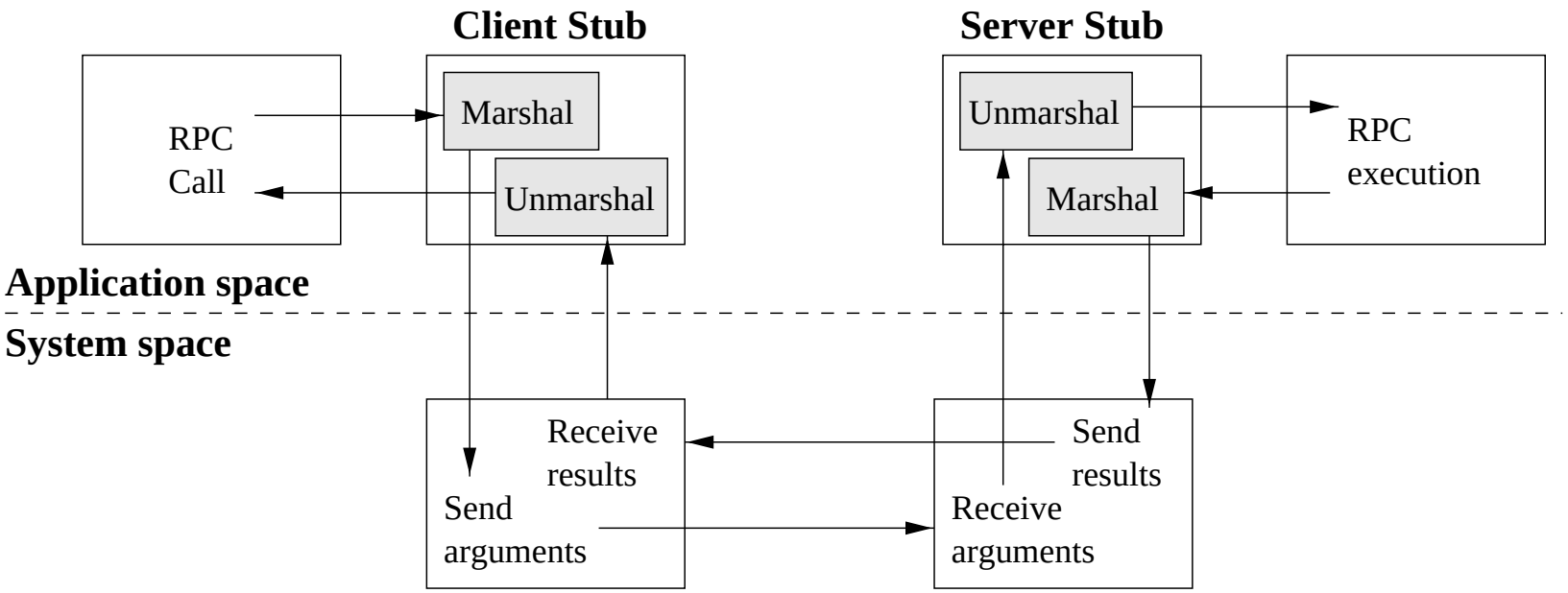
- A realistic experiment
- It drove the development of Tempo, specializer for C
- It highlights key features of partial evaluators

Overview of the talk

- Sun RPC
- Opportunities for specialization
- Application of Tempo
- Discussion on important features

What is RPC?

RPC makes a remote procedure look like a local one



Motivation

Why optimize the Sun RPC?

- Well recognized standard
  - distributed system services: NFS, NIS
  - distributed computing environments: PVM, Stardust
- Performance is critical
  - manual: error prone, does not scale up
  - re-implementation: not compatible with the standard

  $\Rightarrow$ partial evaluation: reuse of existing code

Architecture

- A set of micro-layers
- Highly generic procedures
- IDL and rpcgen:

```
typedef struct { int int1, int2; } pair;
program RMIN_PROG {
  version RMIN_VERS {
    int RMIN(pair) = 0;
  } = 1;
} = 0x20000007;
```

$\Rightarrow$ invariant for specialization

Example: minimum of two integers (client encoding)

```
arg.int1 = ...
arg.int2 = ...
rmin_1(&arg)
 clnt_call()              // Transport protocol switch
  clntudp_call()          // UDP generic procedure call
   xdr_pair()             // Encode 2 integers (rpcgen)
    xdr_int()             // Integer size switch
     xdr_long()           // Encoding/decoding
      XDR_PUTLONG()       // Output protocol switch
       xdrmem_putlong()   // Write buffer/check overflow
        htonl()           // Big/little endian
```

Opportunities for Specialization

Propagation of Exit Status

```
bool_t xdr_pair(xdrs, objp)
{
  if (!xdr_int(xdrs, &objp->int1)) {
    return (FALSE);
  }
  if (!xdr_int(xdrs, &objp->int2)) {
    return (FALSE);
  }
  return (TRUE);
}
```

STAT DYN

Needed: interprocedural, partially static structures, return sensitivity

Opportunities for Specialization

Elimination of Encoding/Decoding Dispatch

```
bool_t xdr_long(xdrs,lp)
{
  if( xdrs->x_op == XDR_ENCODE )
        return XDR_PUTLONG(xdrs,lp);
  if( xdrs->x_op == XDR_DECODE )
        return XDR_GETLONG(xdrs,lp);
  if( xdrs->x_op == XDR_FREE )
        return TRUE;
  return FALSE;
}
```

STAT DYN

Needed: interprocedural, partially static structures, return sensitivity

Opportunities for Specialization

Elimination of Buffer Overflow Checking

```
bool_t xdrmem_putlong(xdrs,lp)
{
  if((xdrs->x_handy -= sizeof(long)) < 0)
    return FALSE;
  *(xdrs->x_private) = htonl(*lp);
  xdrs->x_private += sizeof(long);
  return TRUE;
}
```

STAT DYN

Needed: interprocedural, partially static structures, return sensitivity, use sensitivity (different uses may have different binding times)

Specialized Arguments Encoding (sugared)

```
void xdr_pair(xdrs,objp)
{
  *(xdrs->x_private) = objp->int1;
  xdrs->x_private += 4u;
  *(xdrs->x_private) = objp->int2;
  xdrs->x_private += 4u;
}
```

Manual Interventions

- Control flow for exceptions
- Exposition of specialization opportunities

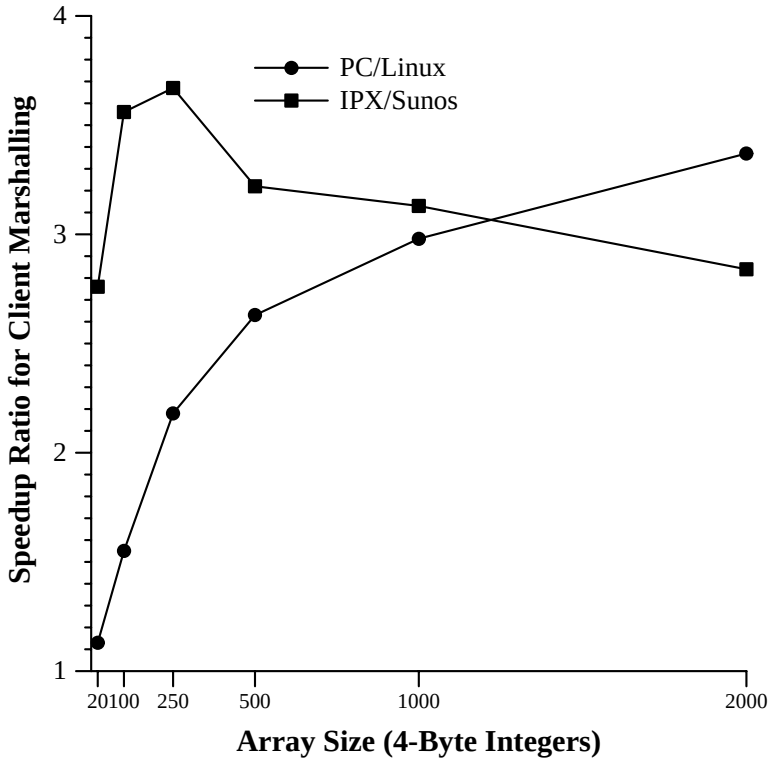
```
inlen = dyn;                    if (inlen == expected) {
code(inlen);        ⟹                inlen = expected;
                                      code(inlen);
                                } else
                                      code(inlen);
```

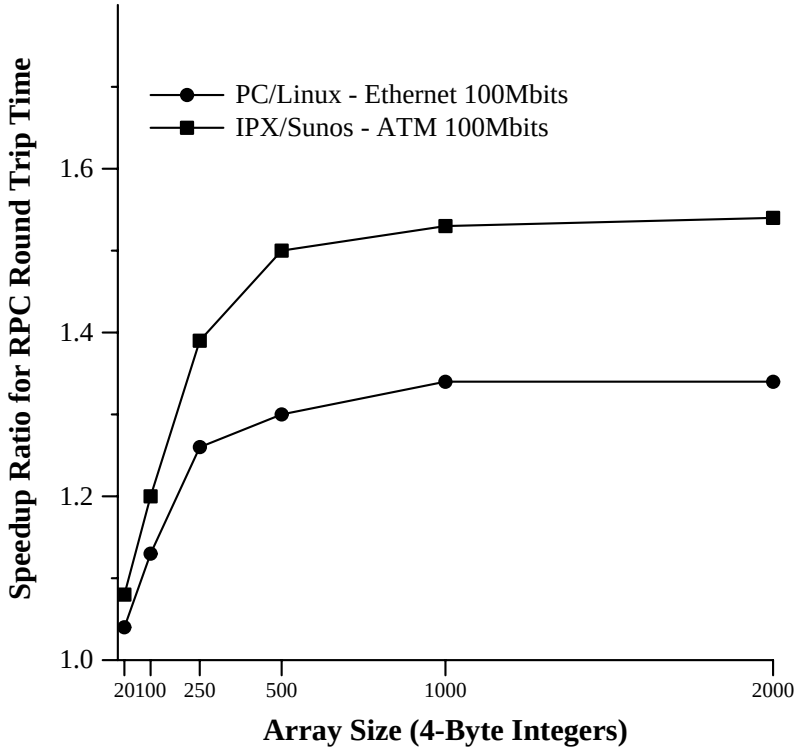
Needed: flow sensitivity

STAT DYN

Application of Tempo

Speedups





Key Features in Binding-Time Analyses

Essential for specializing the Sun RPC:

- Interprocedural analyses
- Partially static structures
- Use sensitivity
- Return sensitivity
- Flow sensitivity

Useful but not essential:

- Context sensitivity

Discussion

User Interface

Crucial to the “tuning” phase:

- Colors for binding times and program transformations
- Aliases
- Polyvariance
- Use of global variables

Useful features:

- Analysis and specialization context

Drawback:

- Code transformation (SUIF, Tempo)

Discussion

Module-Oriented Specialization

- Analysis context prior to code to specialize
- Analysis context after to code to specialize
- Evaluation of external calls
- Abstract description of external functions
  - binding times
  - aliases

Conclusion

- Large scale experiment on existing, mature, commercial code
- Significant speedups
- Key features in BTA
- Key features in user interface
- PE for suppressing modularity and genericity overhead