Mapping Software Architectures to Efficient Implementations via Partial Evaluation

Renaud Marlet, Scott Thibault, Charles Consel

IRISA / INRIA - Université de Rennes 1

Adaptability vs. Speed

?

Contributions

- **Identification**
  of inefficiencies in software architectures
- **Proposition**
  of a systematic optimization technique
- **Validation**
  through case studies

Example: Math Library
Inner Product

```
typedef struct vec_s {
  int      dim;
  double[] ve;
} VEC;

norm  = v_get(3);
light = v_get(3);
...
n_dot_l = _in_prod(norm, light, 0);
```


Safe & Generic Interface

```
double _in_prod(VEC *a, VEC *b, int i0)
{
  if( a == (VEC *)NULL || b == (VEC *)NULL )
    error(E_NULL, "_in_prod");

  limit = min(a->dim, b->dim);
  if( i0 > limit )
    error(E_BOUNDS, "_in_prod");

  return __ip__(&a->ve[i0], &b->ve[i0], limit-i0);
}
```

Safe Generic Safe Layered Generic

Unsafe & Fast (?) Sub-layer

```
double __ip__(double *dp1, double *dp2, int len)
{
  double sum = 0.0;
  for ( i = 0; i < len; i++ )
    sum += dp1[i] * dp2[i];
  return sum;
}
```

Fast Generic Unsafe

Adaptability vs. Speed

```
n_dot_l = _in_prod(norm, light, 0);
```

} Slow Safe Adaptable

```
n_dot_l = norm->ve[0] * light->ve[0] +
          norm->ve[1] * light->ve[1] +
          norm->ve[2] * light->ve[2];
```

} Fast Unsafe Fixed

Why does flexibility introduce overheads ?

Flexibility is not only present in design but also in the implementation.

Alternatives

- Manual rewriting
- Code generators & Scalable libraries
- Automatic program specialization

  → Partial Evaluation

Information Propagation (1)

```
/* LEGEND:  KNOWN  PARTIALLY KNOWN  UNKNOWN  */

typedef struct vec_s {
  int      dim;
  double[] ve;
} VEC;

norm  = v_get(3);
light = v_get(3);
...
n_dot_l = _in_prod(norm, light, 0);
```


Information Propagation (2)

```
/* LEGEND:  KNOWN  PARTIALLY KNOWN  UNKNOWN  */

double _in_prod(VEC *a, VEC *b, int i0)
{
  if( a == (VEC *)NULL || b == (VEC *)NULL )
    error(E_NULL, "_in_prod");
  limit = min(a->dim, b->dim);
  if( i0 > limit )
    error(E_BOUNDS, "_in_prod");
  return __ip__(&a->ve[i0], &b->ve[i0], limit-i0);
}
```


Information Propagation (3)

```
/* LEGEND:  KNOWN  PARTIALLY KNOWN  UNKNOWN  */

double __ip__(double *dp1, double *dp2, int len)
{
  double sum = 0.0;
  for ( i = 0; i < len; i++ )
    sum  += dp1[i] * dp2[i];
  return sum;
}
```

Adaptability *and* Speed

```
n_dot_l = _in_prod(norm, light, 0);
```

Slow
Safe
Adaptable

```
n_dot_l = norm->ve[0] * light->ve[0] +
          norm->ve[1] * light->ve[1] +
          norm->ve[2] * light->ve[2];
```

Fast
Unsafe
Fixed

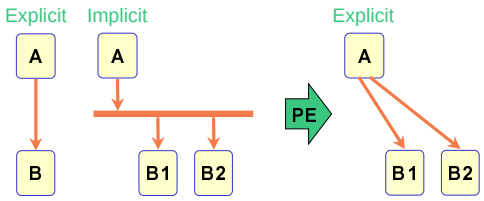
Fast & Safe & Adaptable

Case Studies

Application of Tempo, a partial evaluator for C

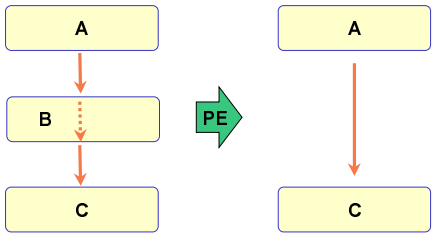
- Selective broadcast (implicit invocation)
- Pattern matching [Field]
- Software layers [Sun RPC]
- Interpreter [Toolbus, DSL]
- Generic library [Meschach]

Invocation



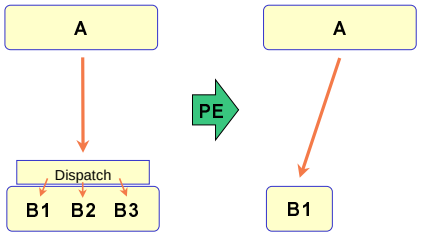
Procedure call
Method invocation
Message broadcast
Parameterized service

Layer Traversal



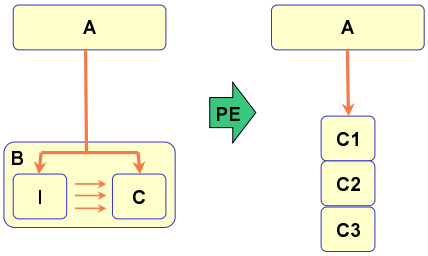
Control Encoded in Data:

Genericity & Parameterization

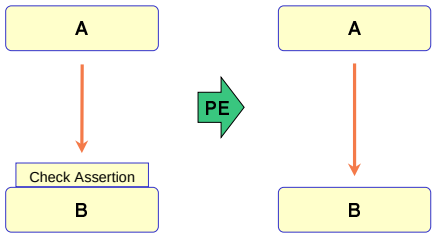


Control Encoded in Data:

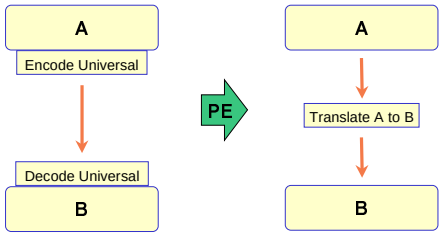
Interpretation



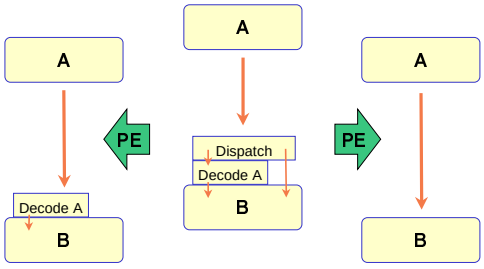
Safety / Security



Data Heterogeneity



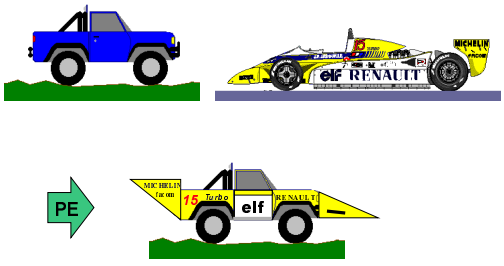
Data Heterogeneity



Sources of Inefficiency in Software Architectures

- Control integration: *invoke what?*
  - Find invoked component(s)
  - Layer traversal
  - Control encoded in data
- Data integration: *use raw data?*
  - Safety / Security
  - Heterogeneity

Adaptability vs. Speed



Future Work

- More case studies (design patterns)
- Macro benchmarks
- Deforestation / Copy elimination
- More applications to Domain Specific Languages

COMPOSE Home Page:

`http://www.irisa.fr/compose`