Optimizing Local Memory Allocation and Assignment Through a Decoupled Approach

Boubacar Diouf [1], Ozcan Ozturk [2], Albert Cohen [1]

[1]INRIA Saclay-Île de France & Université Paris-Sud 11

[2]Bilkent University

October 11, 2009 / LCPC'09

Outline

1. Introduction
2. Decoupled LM Management
3. Experiments
4. Conclusion

- Many processors have Local Memories
  - Digital Signal processors
  - Stream-processing unit (GPUs) and network processors
  - Cell broadband engine's synergetic processing units (SPU)
- Why?
  - Fast
  - Predicatability
  - Power efficiency
- Array allocation? on Local Memory (LM)
  - Allocation decision fixed for the entire execution (static)
  - Allocation depends on the program points (dynamic)

- Many processors have Local Memories
  - Digital Signal processors
  - Stream-processing unit (GPUs) and network processors
  - Cell broadband engine's synergetic processing units (SPU)
- Why?
  - Fast
  - Predicatability
  - Power efficiency
- Array allocation? on Local Memory (LM)
  - Allocation decision fixed for the entire execution (static)
  - Allocation depends on the program points (dynamic)

Motivation

Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial

Motivation

Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial

code

```
d = ...
b = load …
b = b * d
a = load …
a = d / a
c = a / b
a = b + c
store c
store a
```

Motivation

Register Allocation

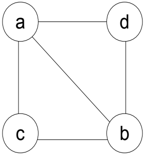
- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial

code

```
d = ...
b = load …
b = b * d
a = load …
a = d / a
c = a / b
a = b + c
store c
store a
```

stack



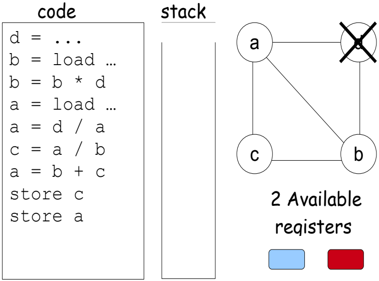
2 Available registers

🟦 🟥

Motivation

Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)



```
d = ...
b = load …
b = b * d
a = load …
a = d / a
c = a / b
a = b + c
store c
store a
```

For more, Please attend SSA tutorial

Motivation

Register Allocation

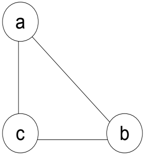
- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial

code

```
d = ...
b = load …
b = b * d
a = load …
a = d / a
c = a / b
a = b + c
store c
store a
```

stack



2 Available registers

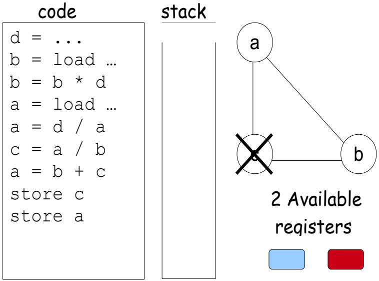
🟦 🟥

Motivation

Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial



```
d = ...
b = load …
b = b * d
a = load …
a = d / a
c = a / b
a = b + c
store c
store a
```

Motivation

Register Allocation

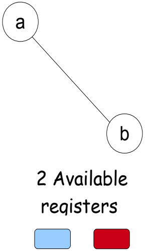
- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial

code

```
d = ...
b = load …
b = b * d
a = load …
a = d / a
c = a / b
a = b + c
store c
store a
```

stack



2 Available registers

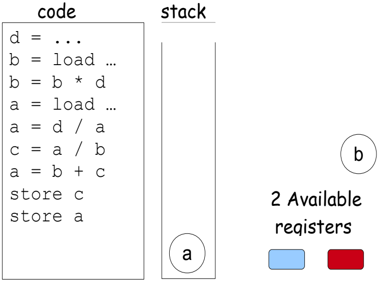
🟦 🟥

Motivation

Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial



code

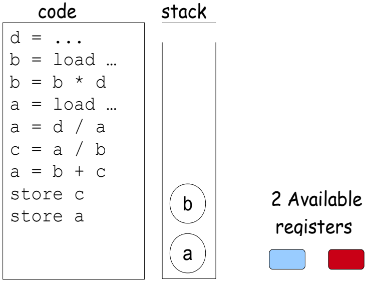
```
d = ...
b = load …
b = b * d
a = load …
a = d / a
c = a / b
a = b + c
store c
store a
```

Motivation

Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial



code

```
d = ...
b = load …
b = b * d
a = load …
a = d / a
c = a / b
a = b + c
store c
store a
```

2 Available registers

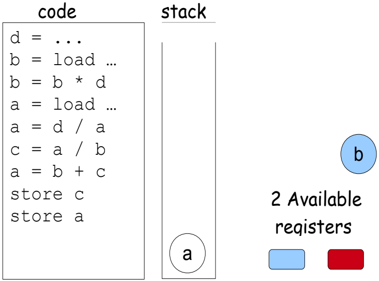
🟦 🟥

Motivation

Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial



code

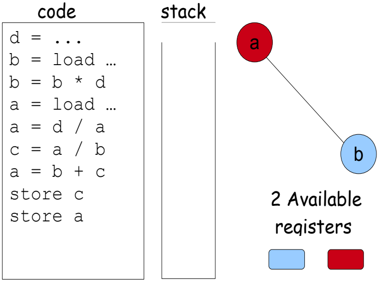
```
d = ...
b = load …
b = b * d
a = load …
a = d / a
c = a / b
a = b + c
store c
store a
```

Motivation

Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial



```
d = ...
b = load …
b = b * d
a = load …
a = d / a
c = a / b
a = b + c
store c
store a
```

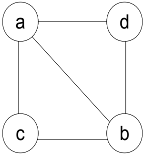
Motivation

Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial

code	live	
d = ...	d	1
b = load …	b,d	2
b = b * d	b,d	2
a = load …	a,b,d	3
a = d / a	a,b	2
c = a / b	b,c	2
a = b + c	a,c	2
store c	a	1
store a		



2 Available registers

🟦 🟥

Motivation

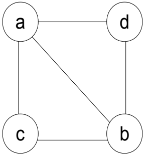
Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial

code	live	
d = ...	d	1
b = load …	b,d	2
b = b * d	b,d	2
a = load …	a,b,d	**3**
a = d / a	a,b	2
c = a / b	b,c	2
a = b + c	a,c	2
store c	a	1
store a		

maxlive



2 Available registers

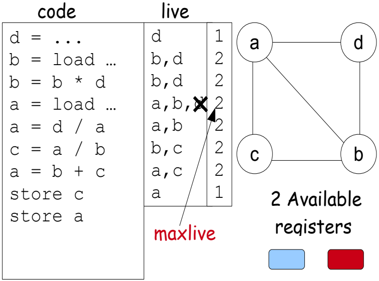
🟦 🟥

Motivation

Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial



code	live	
d = ...	d	1
b = load …	b,d	2
b = b * d	b,d	2
a = load …	a,b,~~d~~	2
a = d / a	a,b	2
c = a / b	b,c	2
a = b + c	a,c	2
store c	a	1
store a		

maxlive

2 Available registers

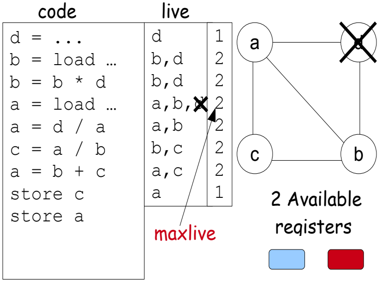
🟦 🟥

Motivation

Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial



code	live	
d = ...	d	1
b = load …	b,d	2
b = b * d	b,d	2
a = load …	a,b,~~d~~	2
a = d / a	a,b	2
c = a / b	b,c	2
a = b + c	a,c	2
store c	a	1
store a		

maxlive

2 Available registers

🟦 🟥

Motivation

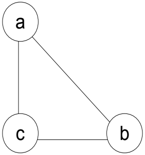
Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial

code	live	
d = ...	d	1
b = load …	b,d	2
b = b * d	b,d	2
a = load …	a,b,~~d~~	2
a = d / a	a,b	2
c = a / b	b,c	2
a = b + c	a,c	2
store c	a	1
store a		

maxlive



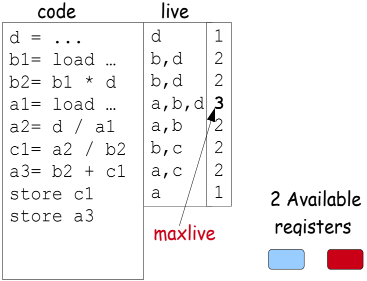
2 Available registers

🟦 🟥

Motivation

Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)



code	live	
d = ...	d	1
b1= load …	b,d	2
b2= b1 * d	b,d	2
a1= load …	a,b,d	3
a2= d / a1	a,b	2
c1= a2 / b2	b,c	2
a3= b2 + c1	a,c	2
store c1	a	1
store a3		

2 Available registers

🟦 🟥

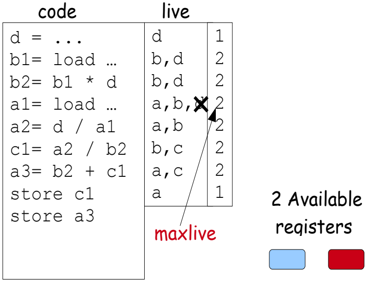
For more, Please attend SSA tutorial

Motivation

Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial



code	live	
d = ...	d	1
b1= load …	b,d	2
b2= b1 * d	b,d	2
a1= load …	a,b,~~d~~	2
a2= d / a1	a,b	2
c1= a2 / b2	b,c	2
a3= b2 + c1	a,c	2
store c1	a	1
store a3		

2 Available registers

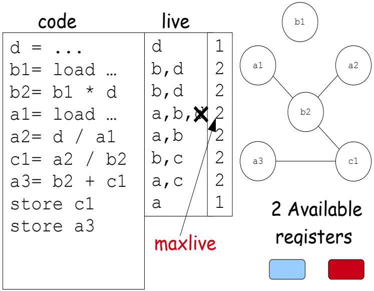
🟦 🟥

Motivation

Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial



code

```
d = ...
b1= load …
b2= b1 * d
a1= load …
a2= d / a1
c1= a2 / b2
a3= b2 + c1
store c1
store a3
```

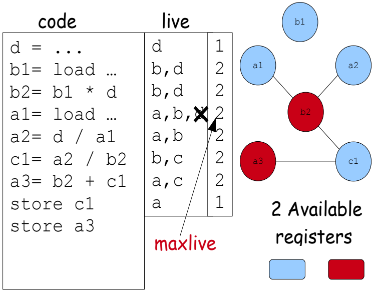
live	
d	1
b,d	2
b,d	2
a,b,~~d~~	2
a,b	2
b,c	2
a,c	2
a	1

Motivation

Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial



code	live	
d = ...	d	1
b1= load …	b,d	2
b2= b1 * d	b,d	2
a1= load …	a,b,~~d~~	2
a2= d / a1	a,b	2
c1= a2 / b2	b,c	2
a3= b2 + c1	a,c	2
store c1	a	1
store a3		

maxlive

2 Available registers

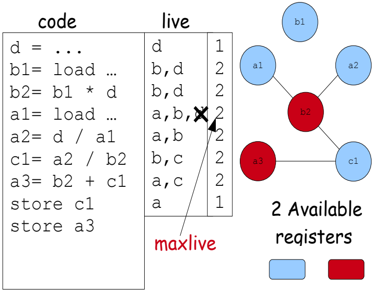
🟦 🟥

Motivation

Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial



code	live	
d = ...	d	1
b1= load …	b,d	2
b2= b1 * d	b,d	2
a1= load …	a,b,~~d~~	2
a2= d / a1	a,b	2
c1= a2 / b2	b,c	2
a3= b2 + c1	a,c	2
store c1	a	1
store a3		

maxlive

2 Available registers

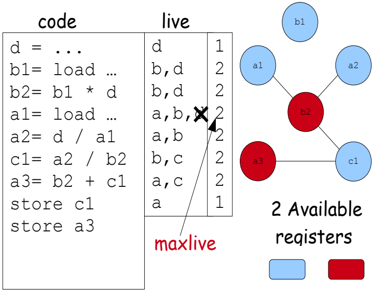
🟦 🟥

Motivation

Register Allocation

- Allocation Phase
  - Rely on Maxlive
  - Choose register residents
- Assignment phase
  - which register for which variable
  - polynomial under SSA
  - Decoupling: isolate the hard problem of allocation (spilling)

For more, Please attend SSA tutorial



code	live	
d = ...	d	1
b1= load …	b,d	2
b2= b1 * d	b,d	2
a1= load …	a,b,~~d~~	2
a2= d / a1	a,b	2
c1= a2 / b2	b,c	2
a3= b2 + c1	a,c	2
store c1	a	1
store a3		

maxlive

2 Available registers

🟦 🟥

Motivation

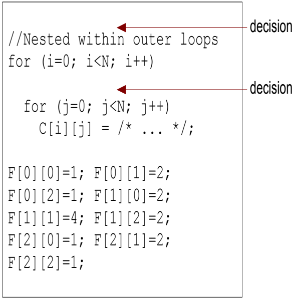
Decoupled Local Memory Allocation

- Allocation Phase
  - Rely on Maxlive, revised as the maximal of the living arrays
  - Choose decision points for splitting
- Assignment phase
  - Which offset for which Array
  - Colorability?
  - Complexity?

Motivation Example

Choice of decision points:

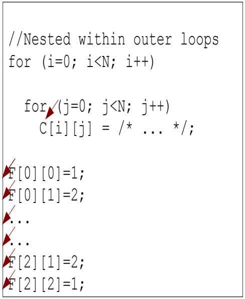
- points where loads and stores are going to be inserted



```
//Nested within outer loops
for (i=0; i<N; i++)

  for (j=0; j<N; j++)
    C[i][j] = /* ... */;

F[0][0]=1; F[0][1]=2;
F[0][2]=1; F[1][0]=2;
F[1][1]=4; F[1][2]=2;
F[2][0]=1; F[2][1]=2;
F[2][2]=1;
```

Allocation schemes

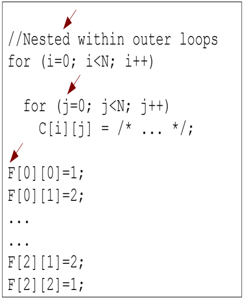
1. At every array instruction
   - finer decision points
   - Excessive Complexity (if ILP used)
2. Every time an array becomes alive
   - Similar to SSA-based register Allocation
3. For the whole method
   - Spill everywhere problem (static)
   - We cannot rely on MaxLive



```
//Nested within outer loops
for (i=0; i<N; i++)

  for (j=0; j<N; j++)
    C[i][j] = /* ... */;

F[0][0]=1;
F[0][1]=2;
...
...
F[2][1]=2;
F[2][2]=1;
```

Allocation schemes

1. At every array instruction
   - finer decision points
   - Excessive Complexity (if ILP used)
2. Every time an array becomes alive
   - Similar to SSA-based register Allocation
3. For the whole method
   - Spill everywhere problem (static)
   - We cannot rely on MaxLive



```
//Nested within outer loops
for (i=0; i<N; i++)

  for (j=0; j<N; j++)
    C[i][j] = /* ... */;

F[0][0]=1;
F[0][1]=2;
...
...
F[2][1]=2;
F[2][2]=1;
```

Allocation schemes

1. At every array instruction
    - finer decision points
    - Excessive Complexity (if ILP used)
2. Every time an array becomes alive
    - Similar to SSA-based register Allocation
3. For the whole method
    - Spill everywhere problem (static)
    - We cannot rely on MAXLIVE

```
//Nested within outer loops
for (i=0; i<N; i++)

  for (j=0; j<N; j++)
    C[i][j] = /* ... */;

F[0][0]=1;
F[0][1]=2;
...
...
F[2][1]=2;
F[2][2]=1;
```

Preliminary transformations

- Tiling
- Loop distribution
- Strip Mining

Preliminary transformations

- Tiling
- Loop distribution
- Strip Mining

Preliminary transformations

- Tiling
- Loop distribution
- Strip Mining

Preliminary transformations

- Tiling
- Loop distribution
- Strip Mining

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    C[i][j] = /* ... */;



F[0][0]=1; /* ... */ F[2][2]=1;
```

Preliminary transformations

- Tiling
- Loop distribution
- Strip Mining

```
for (i=0; i<N; i++)
//Outer strip-mined loop
  for (jj=0; jj<N+B-1; jj+=s)
    // Inner strip-mined loop
    for (j=jj; j<N && j<jj+s; j++)
      C[i][j] = /* ... */;

F[0][0]=1; /* ... */ F[2][2]=1;
```

Preliminary transformations

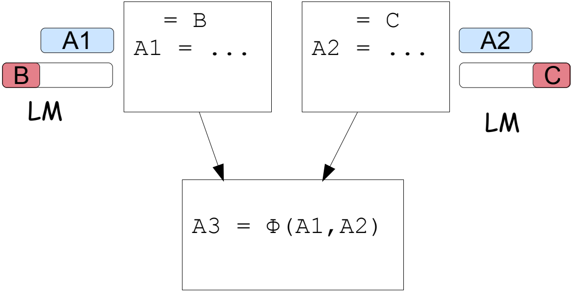
- Tiling
- Loop distribution
- Strip Mining

```
for (i=0; i<N; i++)
  //Outer strip-mined loop
  for (jj=0; jj<N+B-1; jj+=s)
    STORE(C[i][jj..min(jj+B-1,N-1)]);



STORE(F[0..2][0..2]);
```

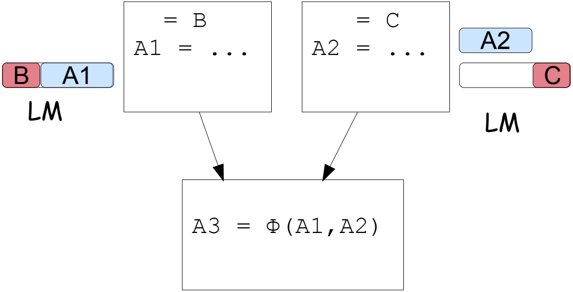
Abstracted Model

- Array blocks are like scalar variables in register allocation
- Extension of SSA to perform on array blocks
  - Not array SSA: no dataflow of individual array elements

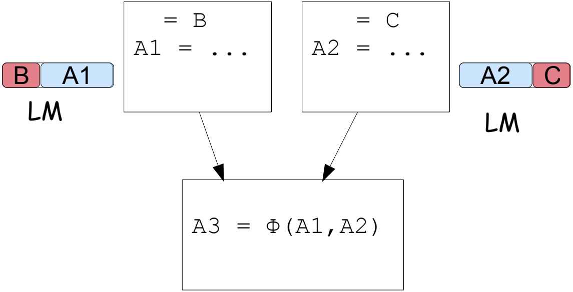
Pointer reconciliation



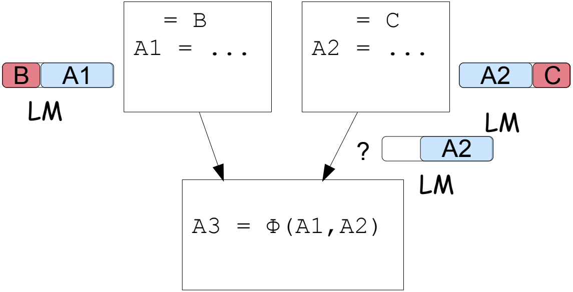
Pointer reconciliation



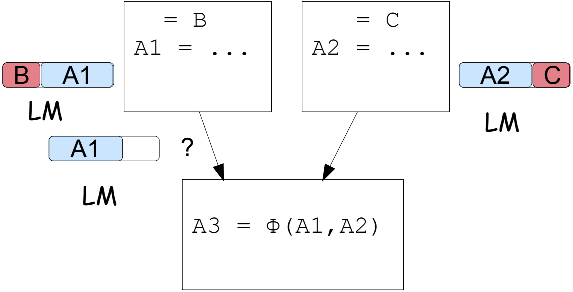
Pointer reconciliation



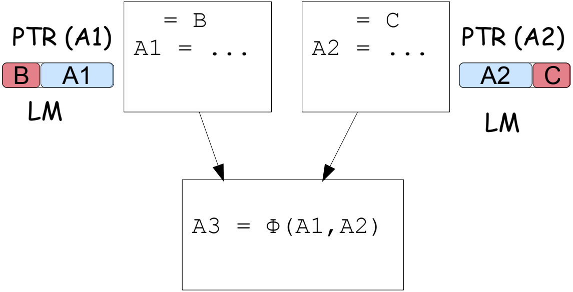
Pointer reconciliation



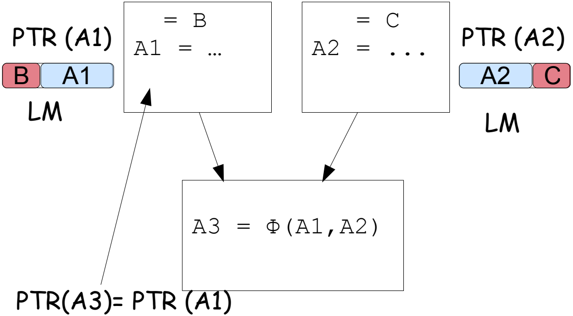
Pointer reconciliation



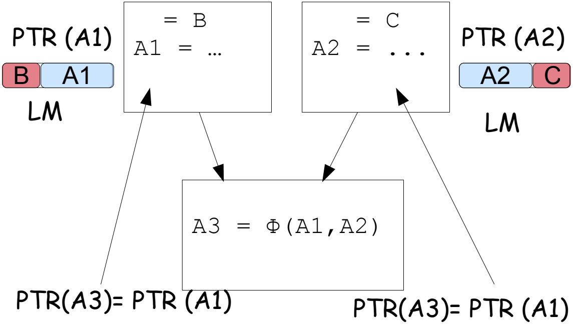
Pointer reconciliation



Pointer reconciliation



Pointer reconciliation



Allocation

- The allocation problem is solved by Integer Linear Programming (ILP)
- Rely on *maxlive* to perform allocation

Assignment

Two options:

- Ignore the fragmentation problem in the allocation step, rely on a fragmentation-avoidance heuristic
- Extend the allocation step to guarantee fragmentation-free assignment: open

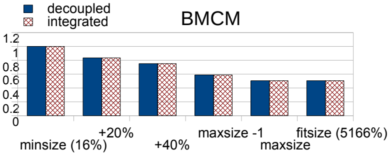
Compare with an integrated approach (not scalable ILP problem)

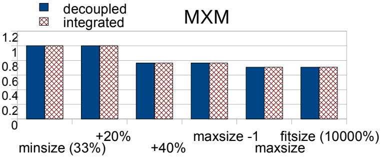
Benchmarks

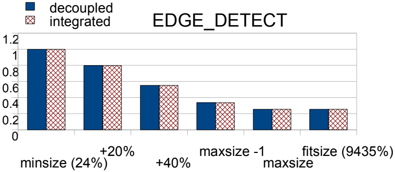
Benchmark	**Brief description**	**Suite**	**Data size**	**arrays /blocks**
`Edge-Detect`	Edge detection in an image	UTDSP	196644	4/385
`D-FFT`	256-point complex FFT	UTDSP	2032	7/7
`Bmcm`	Water molecular dynamics	Perfect Club	125240	10/310
`MxM`	Matrix multiplication	n.a.	120000	3/300

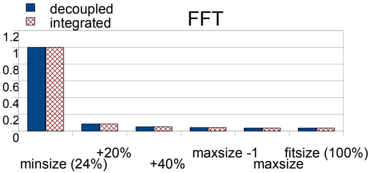
Constant	**Latency**
latency_LM	8
latency_MM	128
$latency\_move(s_v)$	$8+2s_v$
$latency\_spill(s_v)$	$128+4s_v$
$latency\_reload(s_v)$	$128+4s_v$

Results


BMCM


MXM


EDGE_DETECT


FFT

Conclusion

- New bridge between LM management and Register Allocation
- Validation by Experiments
  - Optimal allocation Relying on *maxlive*
  - No fragmentation-induced spills
  - No fragmentation-induced displacements