



SMARQ: Software-Managed Alias Register Queue for Dynamic Optimizations

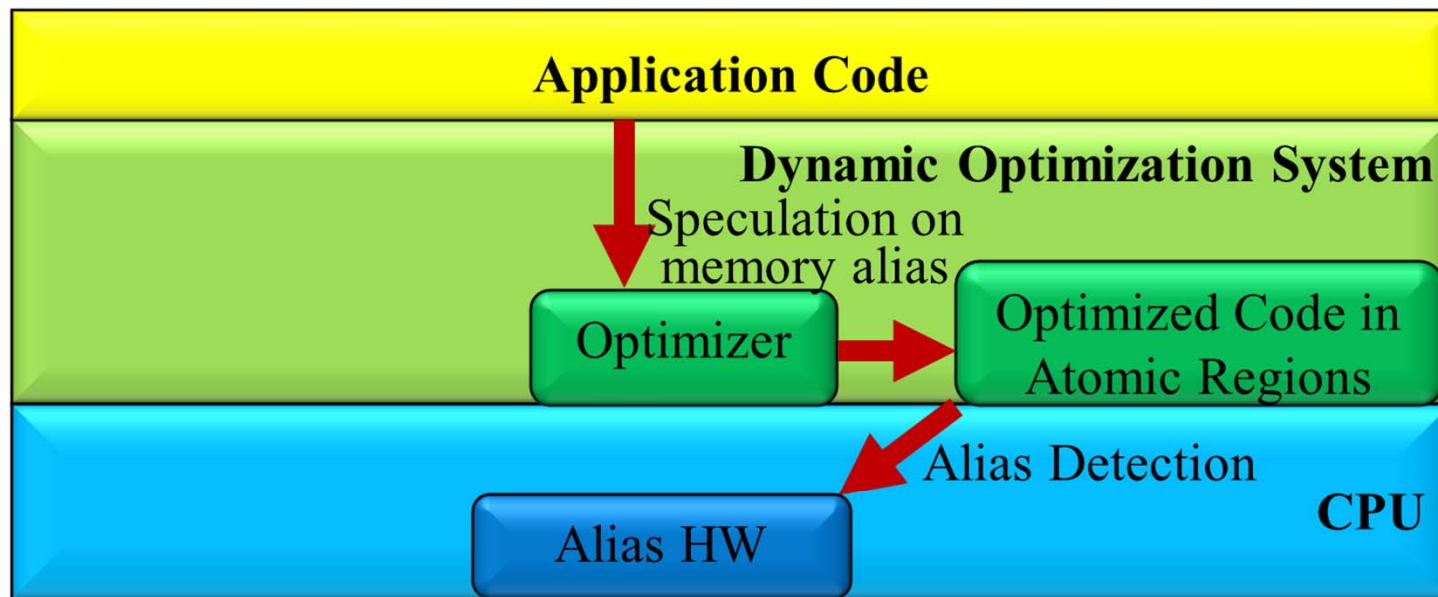
Cheng Wang, Youfeng Wu, Hongbo Rong, Hyunchul Park

Programming Systems Lab
Intel Labs

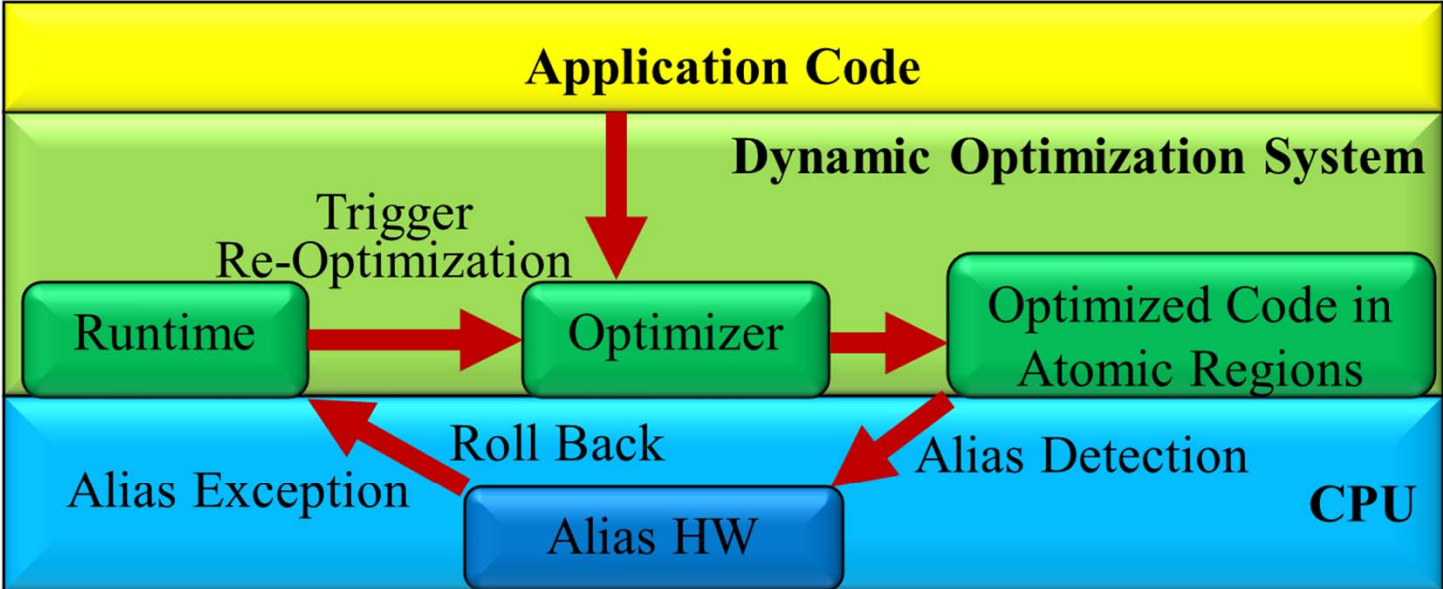
MICRO 2012



Dynamic Optimization with Hardware Alias Detection



Dynamic Optimization with Hardware Alias Detection



Background 1: Hardware Alias Detection

unlikely
alias $\left[\begin{array}{l} M_0: \text{st } [r0] = \dots \\ M_1: \dots = \text{ld } [r1] \end{array} \right.$

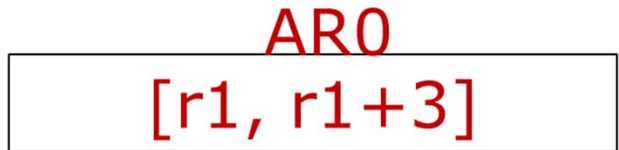
Background 1: Hardware Alias Detection

unlikely alias $\left[\begin{array}{l} M_1: \dots = \text{ld } [r1] \quad , \text{ set alias register AR0} \\ M_0: \text{st } [r0] = \dots \quad , \text{ check alias register AR0} \end{array} \right.$

Background 1: Hardware Alias Detection

SW unlikely alias $\left[\begin{array}{l} M_1: \dots = \text{ld } [r1] \quad , \text{ set alias register AR0} \\ M_0: \text{st } [r0] = \dots \quad , \text{ check alias register AR0} \end{array} \right.$

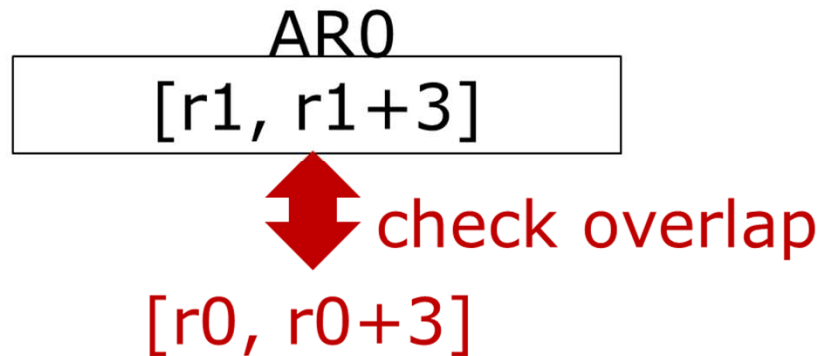
HW



Background 1: Hardware Alias Detection

SW unlikely alias $M_1: \dots = \text{ld } [r1]$, set alias register AR0
 $M_0: \text{st } [r0] = \dots$, check alias register AR0

HW



Multiple Alias Check Issue

M_0 : st [r0+4] = ...
 M_1 : ... = ld [r1]
 M_2 : st [r0] = ...
 M_3 : ... = ld [r2]

Multiple Alias Check Issue

unlikely
alias

M ₃ : ... = ld [r2]	, set AR0
M ₁ : ... = ld [r1]	, set AR1
M ₂ : st [r0] = ...	, check AR0
M ₀ : st [r0+4] = ...	, check AR0, AR1

Multiple Alias Check Issue

unlikely
alias

M ₃ : ... = ld [r2]	, set AR0
M ₁ : ... = ld [r1]	, set AR1
M ₂ : st [r0] = ...	, check AR0
M ₀ : st [r0+4] = ...	, check AR0, AR1

- A memory operation (i.e. M₀) may need to check multiple alias registers (i.e. AR0, AR1)

Multiple Alias Check Issue

unlikely
alias

M ₃ : ... = ld [r2]	, set AR0
M ₁ : ... = ld [r1]	, set AR1
M ₂ : st [r0] = ...	, check AR0
M ₀ : st [r0+4] = ...	, check AR0, AR1

- A memory operation (i.e. M₀) may need to check multiple alias registers (i.e. AR0, AR1)
- Transmeta Efficcon uses a *bitmask* in the instruction to specify the individual alias registers to be checked
 - cannot scale up to a large number of alias registers

Multiple Alias Check Issue

unlikely
alias

M ₃ : ... = ld [r2]	, set AR0
M ₁ : ... = ld [r1]	, set AR1
M ₂ : st [r0] = ...	, check AR0
M ₀ : st [r0+4] = ...	, check AR0, AR1

- A memory operation (i.e. M₀) may need to check multiple alias registers (i.e. AR0, AR1)
- Transmeta Efficeon uses a *bitmask* in the instruction to specify the individual alias registers to be checked
 - cannot scale up to a large number of alias registers
- Itanium Advanced Load Address Table (ALAT) checks *all alias registers* set by previous advanced loads
 - may lead to false positive (i.e. M₂ checks M₁)
 - can not check alias between stores

Background 2: Order-Based Alias Detection

M_0 : st [r0+4] = ...

M_1 : ... = ld [r1]

M_2 : st [r0] = ...

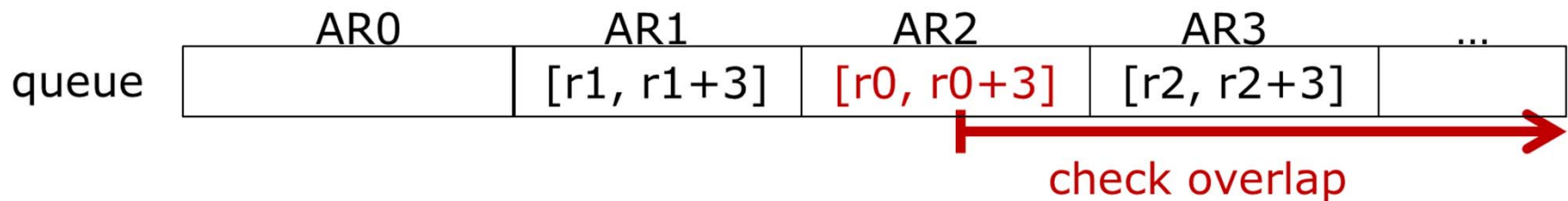
M_3 : ... = ld [r2]

Background 2: Order-Based Alias Detection

M_3 : ... = ld [r2] , order 3
 M_1 : ... = ld [r1] , order 1
 M_2 : st [r0] = ... , order 2
 M_0 : st [r0+4] = ... , order 0

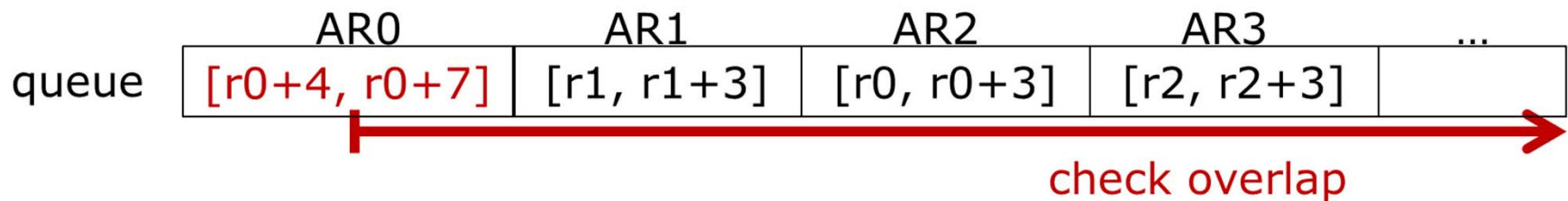
Background 2: Order-Based Alias Detection

```
M3: ... = ld [r2]      , order 3      // set AR3
M1: ... = ld [r1]      , order 1      // set AR1
M2: st [r0] = ...     , order 2      // set AR2, check AR3
M0: st [r0+4] = ...   , order 0
```



Background 2: Order-Based Alias Detection

```
M3: ... = ld [r2]      , order 3      // set AR3
M1: ... = ld [r1]      , order 1      // set AR1
M2: st [r0] = ...     , order 2      // set AR2, check AR3
M0: st [r0+4] = ...   , order 0      // set AR0, check AR1, AR2, AR3
```



Background 2: Order-Based Alias Detection

```

M3: ... = ld [r2]      , order 3      // set AR3
M1: ... = ld [r1]      , order 1      // set AR1
M2: st [r0] = ...     , order 2      // set AR2, check AR3
M0: st [r0+4] = ...   , order 0      // set AR0, check AR1, AR2, AR3
  
```

	AR0	AR1	AR2	AR3	...
queue	[r0+4, r0+7]	[r1, r1+3]	[r0, r0+3]	[r2, r2+3]	

- Adopted in out-of-order CPU (ld/st queue) for HW speculation
- Detect all the aliases between reordered memory operations without any false positive
 - HW automatically identifies loads to prevent alias check between them (E.g. M₁ does not check AR3)

Order-Based Alias Detection Issues

```
M3: ... = ld [r2]      , order 3      // set AR3
M1: ... = ld [r1]      , order 1      // set AR1
M2: st [r0] = ...     , order 2      // set AR2, check AR3
M0: st [r0+4] = ...   , order 0      // set AR0, check AR1, AR2, AR3
```

- Do not leverage compiler analysis for efficient alias register usage
 - 4 alias registers are used
 - M₀ does not need to check M₂

Order-Based Alias Detection Issues

```
M3: ... = ld [r2]      , order 3      // set AR3
M1: ... = ld [r1]      , order 1      // set AR1
M2: st [r0] = ...      , order 2      // set AR2, check AR3
M0: st [r0+4] = ...    , order 0      // set AR0, check AR1, AR2, AR3
```

- Do not leverage compiler analysis for efficient alias register usage
 - 4 alias registers are used
 - M₀ does not need to check M₂
- Only consider memory reordering, but not general optimizations such as load/store elimination
 - May need alias check between non-reordered memory operations

Check Between Non-Reordered Memory Operations

M₀: r3 = **ld** [r0]
M₁: ... = ld [r1]
M₂: **st** [r2] = ...
M₃: r4 = **ld** [r0]

load elimination


[M₀: r3 = **ld** [r0]
M₁: ... = ld [r1]
M₂: **st** [r2] = ...
M₃: r4 = **r3**

M₀: **st** [r0] = r3
M₁: ... = **ld** [r1]
M₂: st [r2] = ...
M₃: **st** [r0] = r4

store elimination


[M₀:
M₁: ... = **ld** [r1]
M₂: st [r2] = ...
M₃: **st** [r0] = r4

Motivation

SW-Managed Alias Register File

- SW Speculation on In-Order CPU
- Itanium - False Positive Issue
- Transmeta - Scalability Issue

HW-Managed Alias Register Queue

- HW Speculation on Out-of-Order CPU
- Efficiency Issue
- Generalization Issue

SMARQ: SW-Managed Alias Register Queue

- Novel architecture features and compiler algorithms
- Solve all the issues in previous works

SMARQ Example


M_0 : st [r0+4] = ...

M_1 : ... = ld [r1]

M_2 : st [r0] = ...


M_3 : ... = ld [r2]

SMARQ Example

 M₃: ... = ld [r2]
M₁: ... = ld [r1]
M₂: st [r0] = ...
M₀: st [r0+4] = ...


- Compiler analysis to derive necessary alias checking for the optimization
 - Check-constraint $A \rightarrow B$: A needs to check B for alias

SMARQ Example

 M₃: ... = ld [r2] , P
M₁: ... = ld [r1] , P
M₂: st [r0] = ... , C
M₀: st [r0+4] = ... , C

- Compiler analysis to derive necessary alias checking for the optimization
 - Check-constraint $A \rightarrow B$: A needs to check B for alias
- Leverage architecture features for efficient management of the alias register queue
 - P bit for setting alias register and a C bit for checking alias registers (may have both)

SMARQ Example

 M₃: ... = ld [r2] , P , order 1
M₁: ... = ld [r1] , P , order 0
M₂: st [r0] = ... , C , order 1
M₀: st [r0+4] = ... , C , order 0

- Compiler analysis to derive necessary alias checking for the optimization
 - Check-constraint $A \rightarrow B$: A needs to check B for alias
- Leverage architecture features for efficient management of the alias register queue
 - P bit for setting alias register and a C bit for checking alias registers (may have both)
- Alias register order allocation respecting check-constraints
 - $A \rightarrow B \Rightarrow \text{order}(A) \leq \text{order}(B)$
 - The alias register order for different instructions with P bit must be different

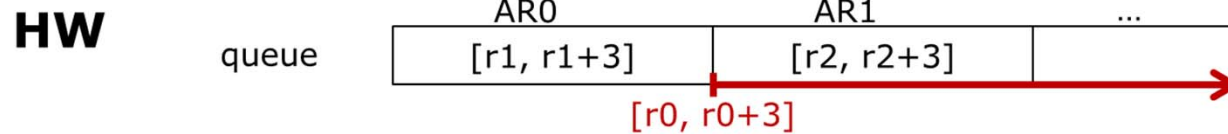
SMARQ Example

SW

```

M3: ... = ld [r2]      , P , order 1      // set AR1
M1: ... = ld [r1]      , P , order 0      // set AR0
M2: st [r0] = ...     , C , order 1      // check AR1
M0: st [r0+4] = ...   , C , order 0

```



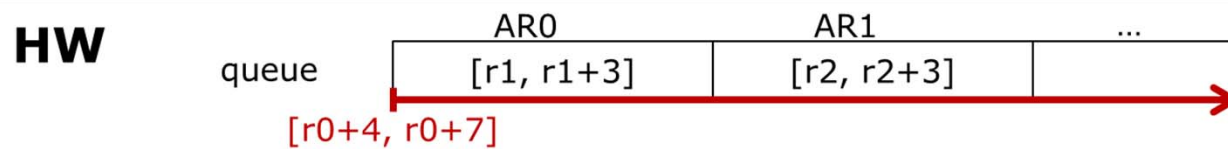
- Compiler analysis to derive necessary alias checking for the optimization
 - Check-constraint $A \rightarrow B$: A needs to check B for alias
- Leverage architecture features for efficient management of the alias register queue
 - P bit for setting alias register and a C bit for checking alias registers (may have both)
- Alias register order allocation respecting check-constraints
 - $A \rightarrow B \Rightarrow \text{order}(A) \leq \text{order}(B)$
 - The alias register order for different instructions with P bit must be different

SMARQ Example

SW

```

M3: ... = ld [r2]      , P , order 1      // set AR1
M1: ... = ld [r1]      , P , order 0      // set AR0
M2: st [r0] = ...     , C , order 1      // check AR1
M0: st [r0+4] = ...   , C , order 0      // check AR0, AR1
    
```



- Compiler analysis to derive necessary alias checking for the optimization
 - Check-constraint $A \rightarrow B$: A needs to check B for alias
- Leverage architecture features for efficient management of the alias register queue
 - P bit for setting alias register and a C bit for checking alias registers (may have both)
- Alias register order allocation respecting check-constraints
 - $A \rightarrow B \Rightarrow \text{order}(A) \leq \text{order}(B)$
 - The alias register order for different instructions with P bit must be different

Prevent False Positive

```
→ M3: ... = ld [r2]      , P  , order 1      // set AR1
→ M1: ... = ld [r1]      , P  , order 0      // set AR0
└─ M2: st [r0] = ...     , C  , order 0      // check AR0, AR1
└─ M0: st [r0+4] = ...  , C  , order 0      // check AR0, AR1
```

- Alias register order respecting check-constraint may cause false positive in alias checking
 - No restriction on the alias register order between M₁ and M₂ and Incorrect order between them will cause M₂ to check M₁

Prevent False Positive

```
→ M3: ... = ld [r2]      , P , order 1      // set AR1
→ M1: ... = ld [r1]      , P , order 0      // set AR0
M2: st [r0] = ...      , C , order 1      // check AR1
M0: st [r0+4] = ...    , C , order 0      // check AR0, AR1
```

- Alias register order respecting check-constraint may cause false positive in alias checking
 - No restriction on the alias register order between M_1 and M_2 and Incorrect order between them will cause M_2 to check M_1
- Anti-constraint $A \dashrightarrow B$: A should not be checked by B
 - $A \dashrightarrow B \Rightarrow \text{order}(A) < \text{order}(B)$

Advanced SMARQ Features

- Alias register rotation*
 - HW organize the alias registers as a circular queue rotated through a base alias register pointer
 - SW rotate the alias register queue to release dead alias register, just like the release of ld/st queue entry in out-of-order CPU
- Handle order cycles*
 - The alias register order respecting all the check-constraints and anti-constraints may contain cycles
 - No alias register order can detect all the aliases without any false positive
 - New architecture alias-register-move feature to break all the cycles
- Prevent alias register overflow*
 - It is hard to support alias register spill
 - Allocate alias register during list scheduling so that if alias register may overflow, schedule instruction to respect all aliases without using new alias registers

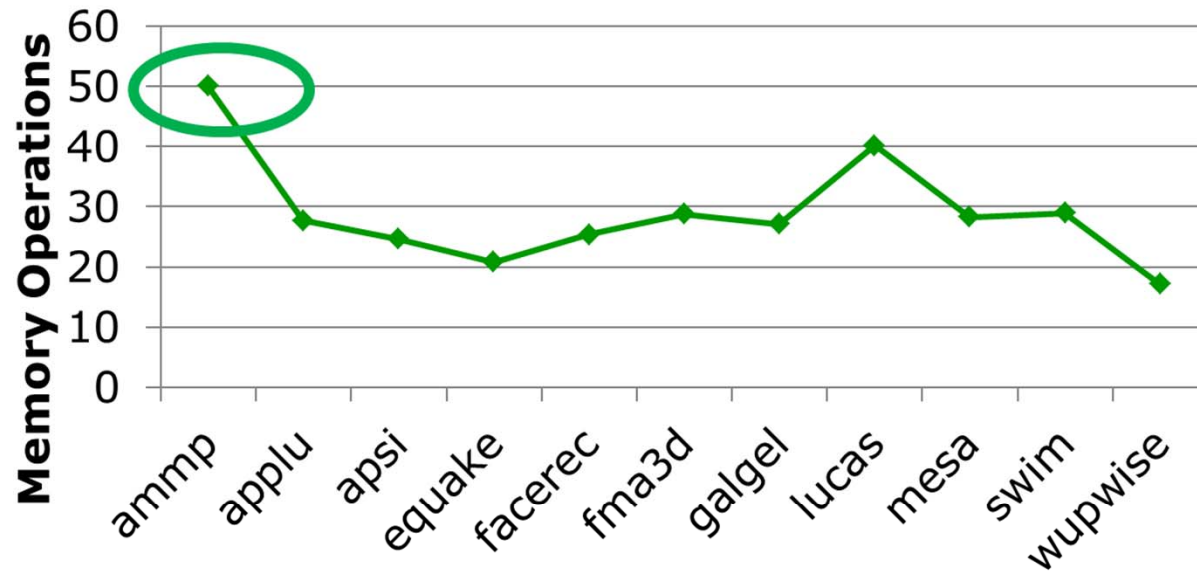
*** See details in the paper**

Experiment Setup

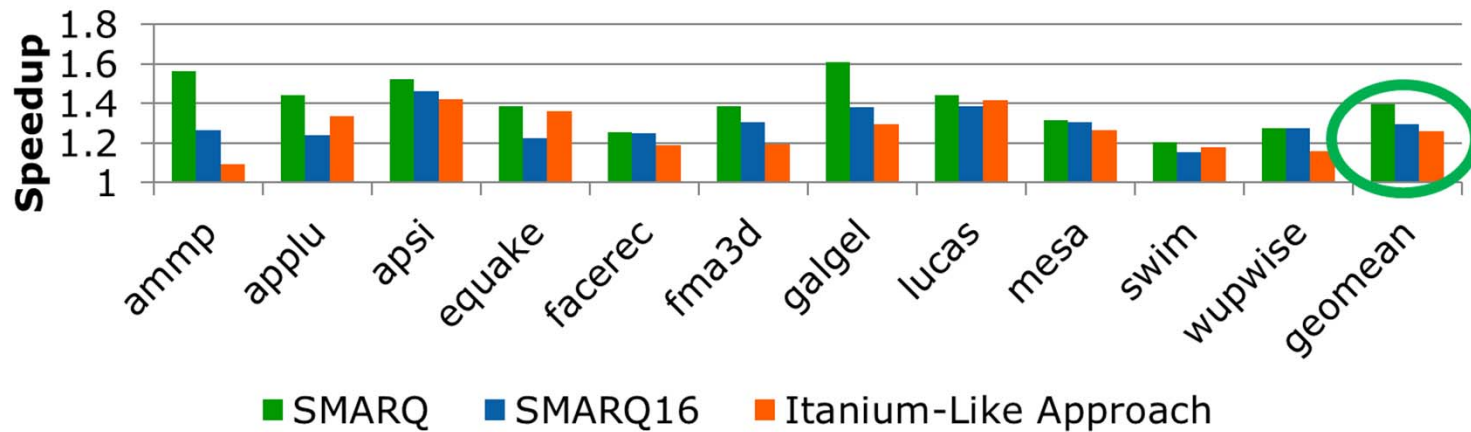
- dynamic optimization framework translates and optimizes x86 binary codes into code running on an internal VLIW CPU modeled by a cycle-accurate

Architecture Features	Parameter
8-wide VLIW	2 INT units, 2 FP units, 2 MEM unit, 1 BRANCH unit, 1 ALIAS unit
L1 I-Cache	4-way 256KB
L1 D-Cache	4-way 64KB, HW prefetch
L2 Cache	8-way 2MB, 8 cycle latency, HW prefetch
L3 Cache	8-way 8MB, 25 cycle latency
Memory	1GB, 104 cycle latency
Alias Register Queue	64 entries

Average Region Size

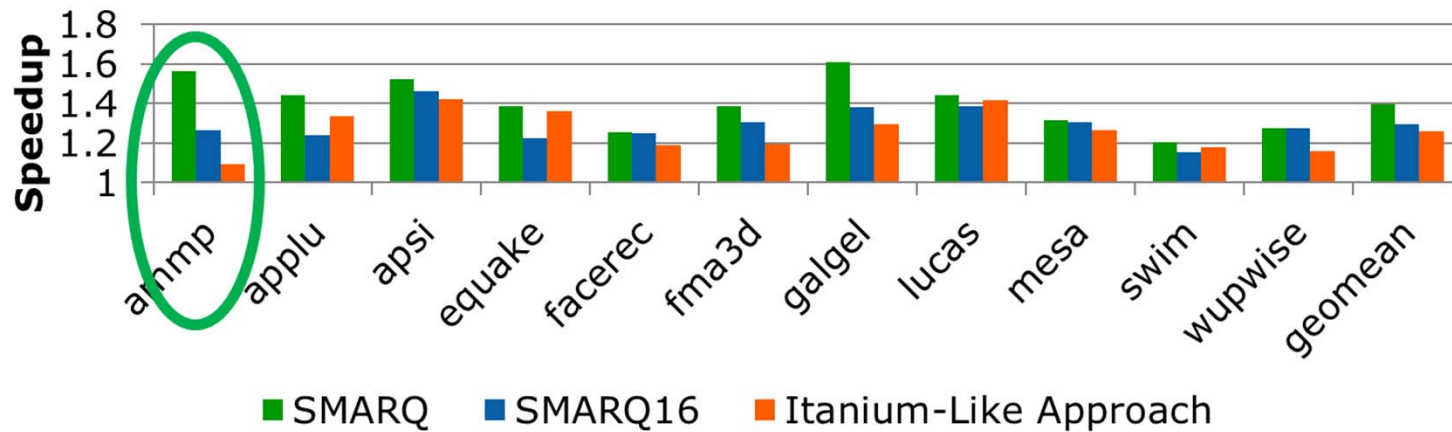


Speedup with Different Alias Detection



- Baseline: no HW alias detection
- SMARQ: 39% speedup on average
- SMARQ16: restrict alias register queue to 16 entries
 - 29% speedup
- Itanium-like approach: non ordered alias detection (with false positives)
 - 26% speedup

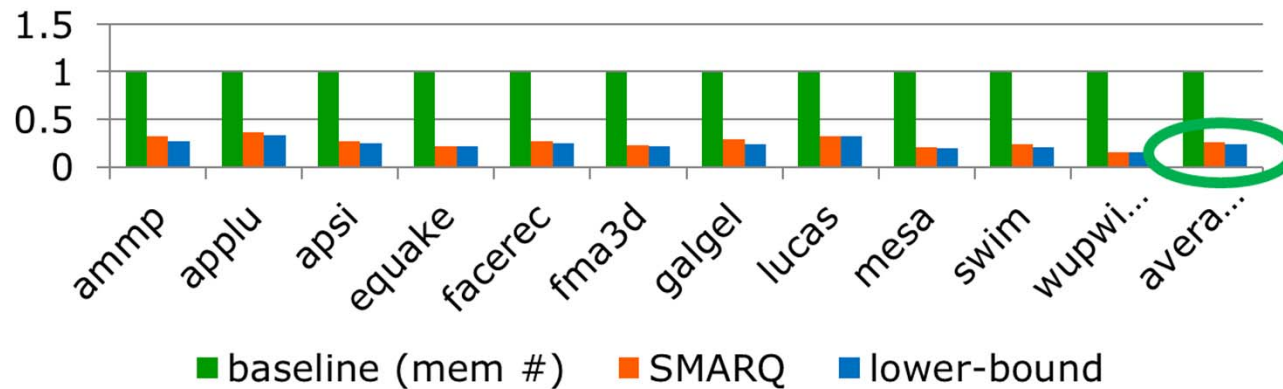
Speedup with Different Alias Detection



- Baseline: no HW alias detection
- SMARQ: 39% speedup on average
- SMARQ16: restrict alias register queue to 16 entries
 - 29% speedup
- Itanium-like approach: non ordered alias detection (with false positives)
 - 26% speedup

Alias Register Working Set

Average Working Set Size



- Baseline: allocate each memory operation a unique alias register in their program order
- SMARQ: reduce the average alias register working set by 74%.
- Lower bound: maximum number of overlapped alias register live-ranges
 - SMARQ achieves alias register working set size close to the lower bound.

Conclusions

- We identify in details the issues in all the previous works on alias registers
- We proposed SMARQ, a software managed alias register queue to solve all the issues in previous works
 - improve the overall performance by 39% as compared to the optimization without alias register
 - reduce the alias register working set by 74%