Compiler Optimisation

12 – Speculative Parallelisation

Hugh Leather
IF 1.18a
hleather@inf.ed.ac.uk

Institute for Computing Systems Architecture
School of Informatics
University of Edinburgh

2019

Introduction

- This lecture on:
  "*LPRD test: Speculative Run-time Parallelisation of loops with privatization and reduction parallelism*"
  - Lawrence Rachwerger PLDI 1995
  - Many follow up papers
  - Expect you to read and understand this paper
- Types of parallel loops
  - Irregular parallelism
  - Reduction parallelism
- LPRD test and examples

Original

```
Do i = 1, N
   A(i)=B(i)
   C(i)=A(i)
Enddo
```

Driver

```
p=get_num_proc()
fork(x_sub,p)
join()
```

Per thread

```
SUBROUTINE x_sub()
  p = get_num_proc()
  z = my_id()
  ilo = N/p * (z-1) +1
  ihi = min(N, ilo+N/p)
  Do i = ilo, ihi
    A(i) = B(i)
    C(i) = A(i)
  Enddo
END
```

Generate *p* independent threads of work

- Each has private local variables, `z`, `ilo`, `ihi`
- Access shared arrays `A`, `B`, and `C`

Privatisation

Original

```
Do i = 1, N
  temp = A(i)
  A(i) = B(i)
  B(i) = temp
Enddo
```

temp privatised

```
Doall i = 1, N
  private temp
  temp = A(i)
  A(i) = B(i)
  B(i) = temp
Enddo
```

- `temp` has loop carried anti and output dependence
- Could scalar expand - but increase storage: $O(1)$ to $O(N)$
- Or private to iteration - storage per processor $O(p), p << N$
- Variable, x, is privatisable for each iteration
  - Every read of x is preceded by write of x

Reduction Parallelism

Original

```
Do i = 1, N
  a = a ⊕ exp
Enddo
```

- Output, flow and anti dependence
- Called a reduction if
  - ⊕ is associative
  - ⊕ is commutative
  - *exp* not contains a

Parallelised

```
pa(z) = 0
Doall i = ilo, ihi
  pa(z) = pa(z) ⊕ exp
Enddo
call barrier_sync()
if(z .EQ. 1)
  Do x = 1, p
    a = a ⊕ pa(x)
  Enddo
Endif
```

- Iteration order does not matter!
- Partial sums in parallel and merge
- Can be sequential $O(p)$ or tree parallel $O(lg\ p)$

Irregular Parallelism

Indirect array accesses

```
Do i = 1 to N
  A(X(i)) = A(Y(i)) + B(i)
Enddo
```

- Loop carried output dependent if any $X(i_1) = X(i_2),\ i_1 \neq i_2$
- Loop carried flow/anti dependent if any $X(i_1) = Y(i_2),\ i_1 \neq i_2$
- Values of X, Y determine dependence
    - Unknown at compile-time
- More than half scientific programs are irregular - sparse arrays

Runtime Parallelisation

Original

```
Do i = 1, N
  A(i+k) = A(i) + B(i)
Enddo
```

No dependence if `|k| > N`

Guarded parallelism

```
If(-N < K < N)
  Do i = 1, N
    A(i+k) = A(i) + B(i)
  Enddo
Else
  Doall i = 1, N
    A(i+k) = A(i) + B(i)
  Enddo
Endif
```

- Multiple versions of code
- Analysis at runtime
- Here check simple but can be more complex

Speculative Parallelisation

Original

```
Do i = 1, N
  A(w(i)) = A(r(i)) + B(i)
Enddo
```

- Assume parallel
- Loop not parallel if any $r(i_1) = w(i_2), i_1 \neq i_2$
- Collect data access pattern and verify if dependence could occur[1]

Speculative

```
cp = checkpoint()
Doall i = 1, N // parallel
  trace_A(w(i), r(i))
  A(w(i)) = A(r(i)) + B(i)
Enddo
fail = analyse()
If (fail) // sequential
  restore(cp)
  DO i = 1, N
    A(w(i)) = A(r(i))+B(i)
  Enddo
Else
  discard(cp)
Endif
```

[1] Compare vs check dependences not violated

Definitions

Independent Shared Variables

```
do i=1,n
  f(i) = A(i)
  B(i) = g(i)
end do
```

a shared variable is independent if it is:

- read-only (e.g., `A`)
- accessed (written and read) in only one iteration (e.g., `B`)

Definitions

Privatisable Shared Variables

```
do i=1,n
  A(l:m) = f(i)
  h(i) = A(l:m)
end do
```

a shared array A can be *privatised* if and only if

- every read access to an element of A is preceded by a write access to that same element of `A` within the same iteration of the loop
- it is dead after the loop

Lazy privatising `Doall` test

- Speculatively privatise array elements and parallelise loop
- Shadow arrays to record array accesses (per processor)
  - If one iteration writes memory and another reads but does not write it – not Doall, speculation failed
  - Else if no memory written by different iterations – is Doall, speculation succeeded
  - Else if any iteration a value is read before it is written – not privatisable, speculation failed
  - Else speculation succeeded!

LRPD test Example

Loop

```
A(4), B(5),K(5), L(5)
Do i = 1, 5
  z = A(K(i))
  If B(i) .EQ. 0 then
    A(L(i)) = z + C(i)
  Endif
Enddo
```

Array contents

```
B(1:5) = (1,0,1,0,1)
K(1:5) = (1,2,3,4,1)
L(1:5) = (2,2,4,4,2)
```

Unsafe if $K(i_1) = L(i_2), B(i_2) = 0, i_1 \neq i_2$

Is it safe?

LRPD test Example

Loop

```
A(4), B(5),K(5), L(5)
Do i = 1, 5
  z = A(K(i))
  If B(i) .EQ. 0 then
    A(L(i)) = z + C(i)
  Endif
Enddo
```

Array contents

```
B(1:5) = (1,0,1,0,1)
K(1:5) = (1,2,3,4,1)
L(1:5) = (2,2,4,4,2)
```

Unsafe if $K(i_1) = L(i_2), B(i_2) = 0, i_1 \neq i_2$

Is it safe?

Only consider i_2 when $B(i_2) = 0$, gives $i_2 \in \{2, 4\}$

LRPD test Example

Loop

```
A(4), B(5),K(5), L(5)
Do i = 1, 5
  z = A(K(i))
  If B(i) .EQ. 0 then
    A(L(i)) = z + C(i)
  Endif
Enddo
```

Array contents

```
B(1:5) = (1,0,1,0,1)
K(1:5) = (1,2,3,4,1)
L(1:5) = (2,2,4,4,2)
```

Unsafe if $K(i_1) = L(i_2), B(i_2) = 0, i_1 \neq i_2$

Is it safe?

Only consider i_2 when $B(i_2) = 0$, gives $i_2 \in \{2, 4\}$

$L(2) = 2, L(4) = 4$, only matches in K when $i_1 = i_2$

LRPD test Example

Loop

```
A(4), B(5),K(5), L(5)
Do i = 1, 5
  z = A(K(i))
  If B(i) .NE. 0 then
    A(L(i)) = z + C(i)
  Endif
Enddo
```

Array contents

```
B(1:5) = (1,0,1,0,1)
K(1:5) = (1,2,3,4,1)
L(1:5) = (2,2,4,4,2)
```

Unsafe if $K(i_1) = L(i_2), B(i_2) = \mathbf{1}, i_1 \neq i_2$

Is it safe?

LRPD test Example

Loop

```
A(4), B(5),K(5), L(5)
Do i = 1, 5
  z = A(K(i))
  If B(i) .NE. 0 then
    A(L(i)) = z + C(i)
  Endif
Enddo
```

Array contents

```
B(1:5) = (1,0,1,0,1)
K(1:5) = (1,2,3,4,1)
L(1:5) = (2,2,4,4,2)
```

Unsafe if $K(i_1) = L(i_2), B(i_2) = \mathbf{1}, i_1 \neq i_2$

Is it safe?

When $i_1 = 2, i_2 = 1$ then

$K(i_1 = 2) = 2 = L(i_2 = 1)$ and $B(i_2 = 1) = 1$

LRPD test Marking phase

- Allocate shadow arrays A_w, A_r, A_{np} one per processor. $O(n \times p)$ overhead. Speculatively privatise `A` and execute in parallel. Record accesses to data under test in shadows
- `markwrite(A(i))`:
  - Increment tw_A (write counter)
  - If first time `A(i)` written in iteration, mark A_w`(i)`, clear A_r`(i)`
  - (Only concerned with cross-iteration dependences)
- `markread(A(i))`:
  - If `A(i)` not already written in iteration, mark A_r`(i)` and mark A_{np}`(i)`
  - Note A_{np}`(i)` not cleared by MarkWrite. np = 'not privatisable if written elsewhere'

LRPD test Marking phase

```
A(4), B(5),K(5), L(5)
Doall i = 1,5
  markread(A(K(i)))
  z = A(K(i))
  If B(i) then
    markwrite(A(L(i)))
    A(L(i)) = z + C(i)
  endif
Enddo
```

Note, some effort to optimise placement of marking.

LRPD test Results after marking

Program

```
A(4), B(5),K(5), L(5)
Do i = 1, 5
  z = A(K(i))
  If B(i) .EQ. 0 then
    A(L(i)) = z + C(i)
  Endif
Enddo
```

Array contents

```
B(1:5) = (1,0,1,0,1)
K(1:5) = (1,2,3,4,1)
L(1:5) = (2,2,4,4,2)
```

LRPD shadows

	1	2	3	4
$A_w(1{:}4)$	0	1	0	1
$A_r(1{:}4)$	1	0	1	0
$A_{np}(1{:}4)$	1	0	1	0
$A_w \wedge A_r$	0	0	0	0
$A_w \wedge A_{np}$	0	0	0	0

$tm_A = \sum A_w = 2$

Total number of distinct elements written

LRPD test Analysis phase

- if $A_w \wedge A_r$ then NOT Doall read and write in diff iterations to same element
- else if $tw = tm$ then was a Doall unique iterator writes
- else if $A_w \wedge A_{np}$ then NOT Doall
- otherwise loop privatisation valid, Doall

$Aw \wedge A_r = 0$: Fail
$tw \neq tm$: Fail
$A_w \wedge A_{np} = 0$: Fail
Overall privatise - remove output dependence

LRPD test Marking phase

Handling reductions

- Extended to handle reductions
- Allocate shadow arrays per processor. $O(n \times p)$ overhead.
- Record accesses to data under test in shadows
- Mark Redux ()
  - Mark `A(i)` if element is NOT valid reference in reduction statement - not a reduction variable
- Read paper for details and example

LRPD test Improvements

- One dependence can invalidate speculative parallelisation
  - Partial parallelism not exploited
  - Transform so that up till first dependence parallel
  - Reapply on the remaining iterators.
- Large overheads
  - Adaptive data structures to reduce shadow array overhead
- Large amount of work in speculative parallelisation
  - Hardware support for Thread Level Speculation (TLS), transactional memory
  - Compiler combined with static analysis

Summary

- Summary of parallelisation idioms
- Irregular accesses
- Shadow arrays
- Marking and analysis for Doall and reductions
- Last lecture on parallelism. Next on adaptive compilation

PPar CDT Advert

EPSRC Centre for Doctoral Training in
Pervasive Parallelism

- 4-year programme:
  MSc by Research + PhD
- Research-focused:
  Work on your thesis topic from the start
- Collaboration between:
  - University of Edinburgh's School of Informatics
    - Ranked top in the UK by 2014 REF
  - Edinburgh Parallel Computing Centre
    - UK's largest supercomputing centre

- Research topics in software, hardware, theory and application of:
  - Parallelism
  - Concurrency
  - Distribution
- Full funding available
- Industrial engagement programme includes internships at leading companies

The biggest revolution in the technological landscape for fifty years

Now accepting applications!
Find out more and apply at:
pervasiveparallelism.inf.ed.ac.uk

THE UNIVERSITY *of* EDINBURGH
informatics

EPSRC
Engineering and Physical Sciences Research Council