MANCHESTER
1824
The University
of Manchester

A join point for loops in AspectJ

Bruno Harbulot and John Gurd

The University of Manchester

FOAL 2005 – Chicago, March 2005

What we would like to do

- Writing aspects that represent the concern:
  - "parallelise all the loops iterating from 0 to the length of an array of int using MPI",
  - or "parallelise all the loops iterating over a Collection using Java Threads".
- Write (aspect) code that does not invade the readability of the numerical code.

Previously, on loops and AspectJ…

- "*Using AspectJ to Separate Concerns In Parallel Scientific Java Code*" (AOSD 2004)
- Parallelisation of loops using aspects:
  - by making the iteration space visible as parameters to the methods
  - by turning loops into self-contained objects (loop body and boundaries)
- Both require refactoring the base code

Presentation Outline

- Join point model:
  - Part 1: Shadows (static part),
  - Part 2: Context exposure (dynamic part),
- Loop selection,
- Implementation using `abc`,
- Dealing with exceptions,
- Related topics.

Join Points

- A **join point** is “*a point in the dynamic call graph of a running program*”.
- A join point **shadow** is its location in the text of the program.
- Ability to weave code *before*, *after* and/or *around*.
- Ability to access **execution context**.

JP Part 1: Shadows (static)

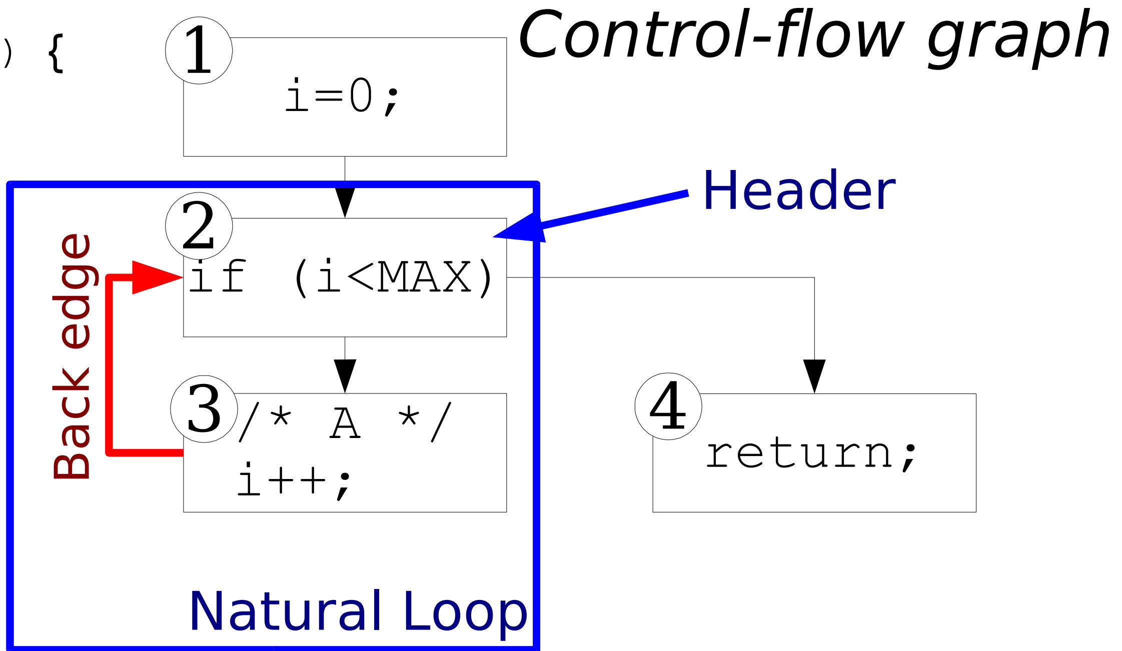
- Analysis of the control flow graph
- Finding natural and combined loops
- Classification of loops according to their weaving and analysis capabilities:
  - General loops
  - Loops with unique successor
  - Loops with unique exit node

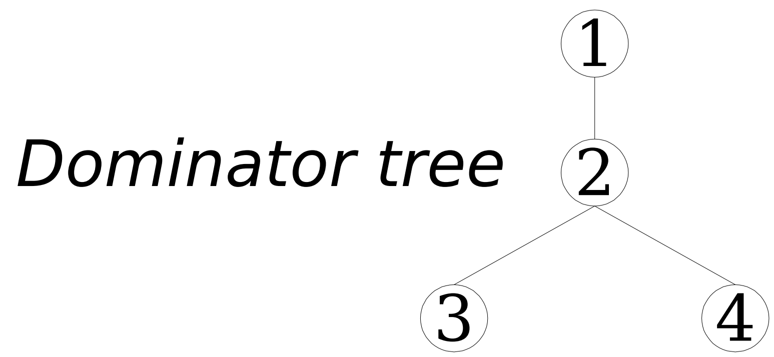
Control-flow graph, dominators and natural loops (I)

- A node is a **basic block** (only entry via its head and only exit via its tail).
- Node *d* **dominates** node *n* if every path from the beginning to *n* goes through *d*.
- A **back edge** (*a* -> *b*) is an edge whose head (*b*) dominates its tail (*a*).
- Given a back edge *n* -> *d*, the natural loop is *d* plus the set of nodes that can reach *n* without going through *d*.

Control-flow graph, dominators and natural loops (II)

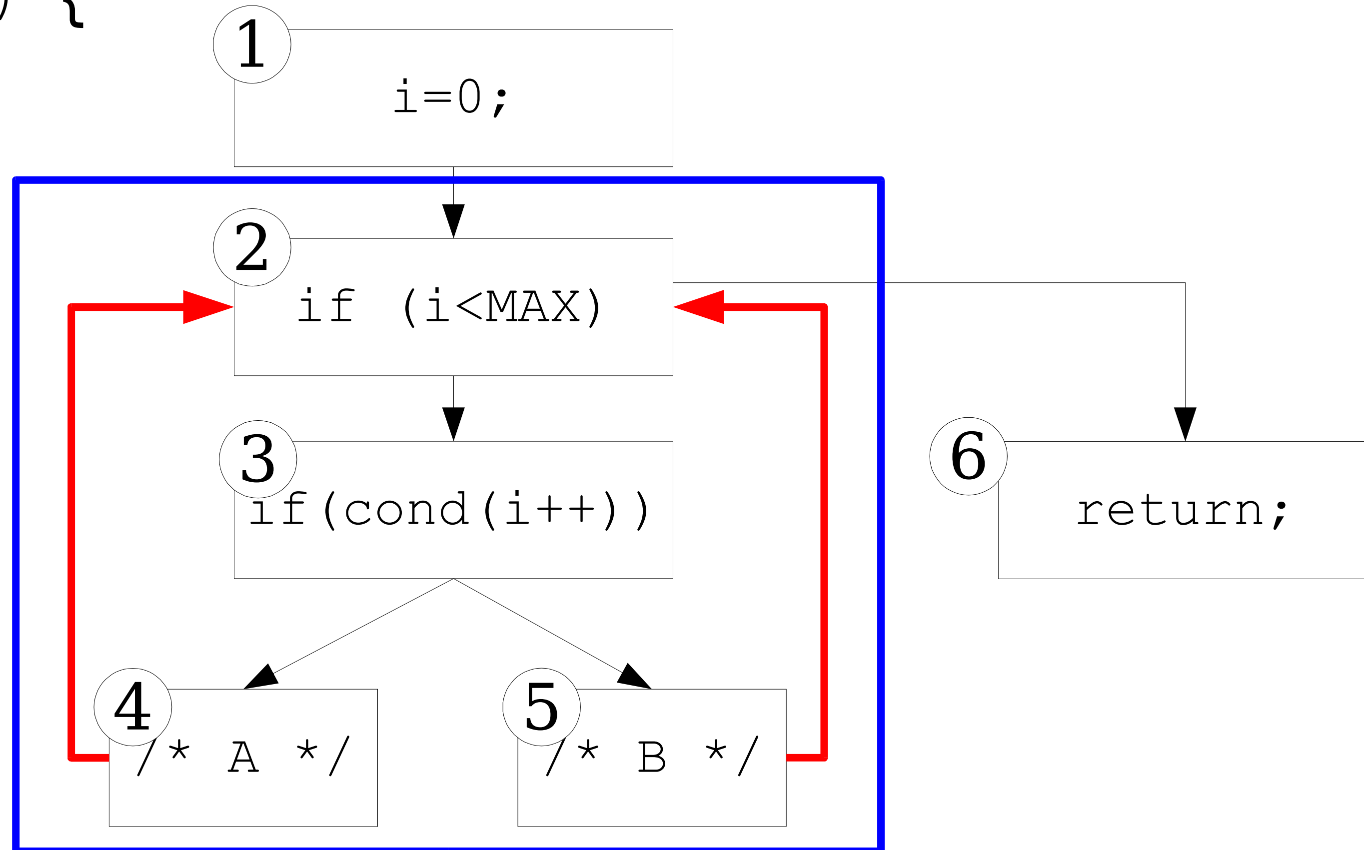
```
for (int i = 0 ; i<MAX ; i ++) {
    /* A */
}

int j = 0 ;
int STRIDE = 1 ;
for (; j < MAX ; j+=STRIDE) {
    /* A */
}

int k = 0 ;
while (k < MAX) {
    /* A */
    k ++ ;
}
```





Combined loops

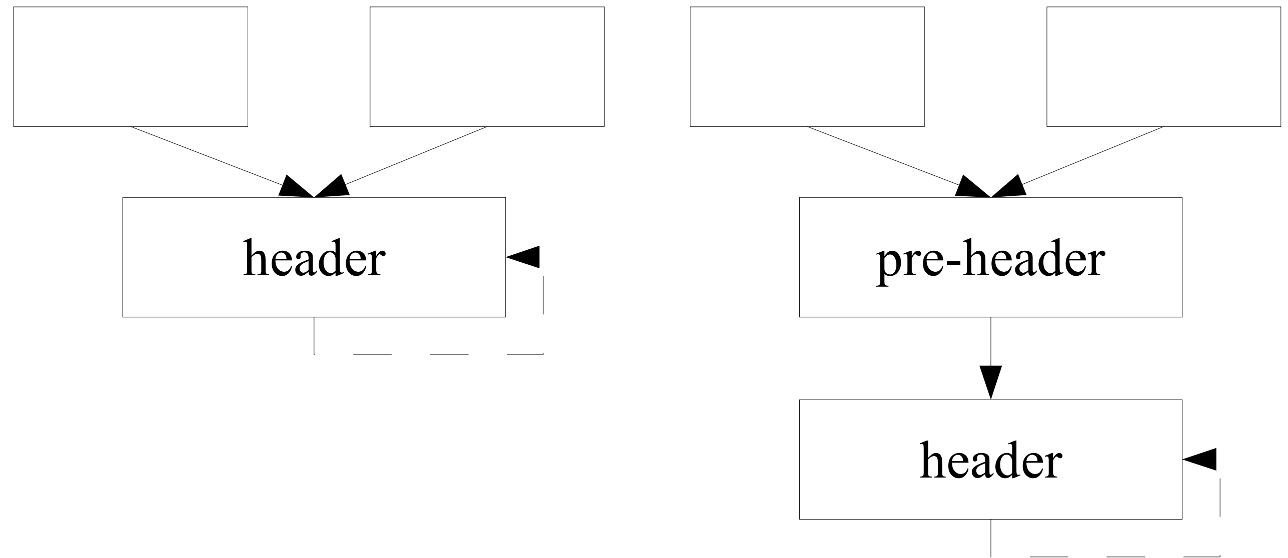
```
int i = 0 ;
while (i< MAX) {
   if (cond(i++)) {
      /* A */
   } else {
      /* B */
   }
}
```



1 combined loop with 2 back edges

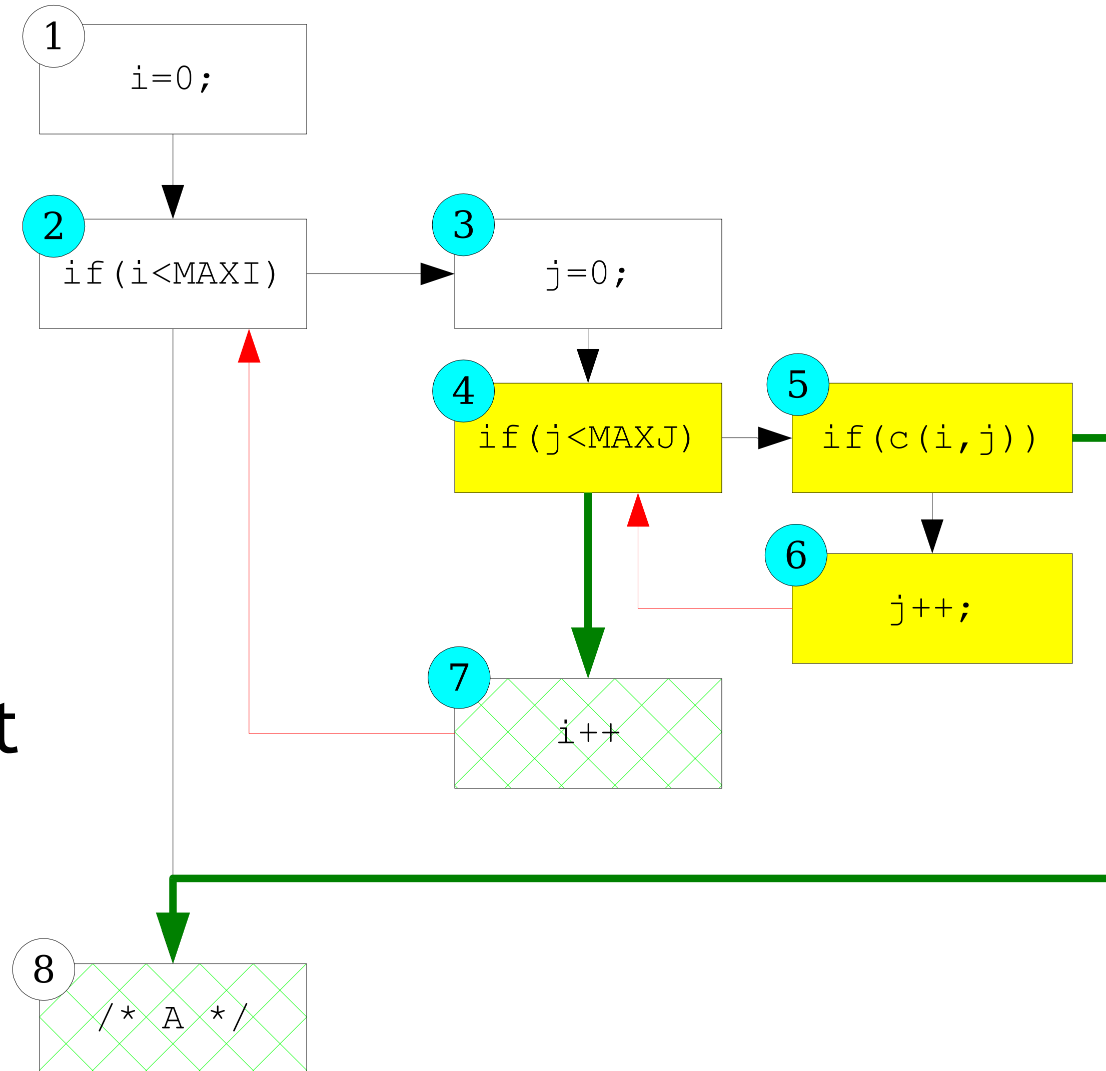
“Before” the loop

- Always possible
- Inserting a pre-header



"After" and "around" the loop

- Unique successor: unique point after (around possible).
- Multiple successors: multiple points after (around impossible).
- Loops with unique exit node allow further behaviour prediction.



JP Part 2: Context Exposure (dynamic)

- Exposing data processed and guiding the execution,
- “Arguments” to the loop,
- Integer range and `Iterator`s,
- Arrays and `Collection`s.
- (Only loop with unique exit nodes to avoid “`break`” statements and irregular iterations)

Context Exposure

- For method calls (for example), the context exposed comprises the target, the caller object and the arguments,
- Need similar data for loops to exploit the loop join point potential,
- Otherwise, only able to recognise that there is a loop, but no extra information on what it does.

Integer range and Iterators

- `for (int i = min ; i < max ; i+=1)`
- Need to get *min*, *max* and *stride* for parallelisation.
- `while (iter.hasNext()){ … iter.next() … }`
- Need to get `Iterator` *iter*.
- Passed as "`args(min, max, stride)`" or "`args(iter)`".

Arrays and Collection

- Analogy with Java 5 (Tiger) constructs.
- `for (Object item: collec) { ... }`
- ```
  Iterator iter = collec.iterator();
  while (iter.hasNext()) {
      Object item = iter.next() ;
      ...
  }
  ```
- Provides extra information about the data processed by the loop.
  ```

Loop selection

- In AspectJ, the selection is (ultimately) based on a name pattern, for example on the method name or an argument type,
- Loops haven't got names,
- Selection to be made on argument types and on data processed: integer range and Iterators; and especially arrays and Collections. (+`cflow`, `within` and `withincode`)
- ```
  pointcut bytearrayloop(int min,int max,int s,byte[] a): loop() && args(min,max,s,a);
  ```
  ```

Implementation using `abc`

- `abc`: AspectBench Compiler (full AspectJ compiler),
- *LoopsAJ*: our extension for `abc` that implements a loop pointcut,
- Analysis capabilities of Soot,
- Need to update the graph when weaving,
- Only one “after” point possible,

Dealing with exceptions

- The graph is not necessarily "reducible" (loops may have several entry points),
- The traps for the exceptions do not necessarily match anything in the source code.

Related topics: loop-body join point

- It would be possible to insert a node similar to the “pre-header”, but for edges from the loop.
- This would comprise the evaluation of the condition within the definition of the “loop-body”.
- What would context could be exposed?

Summary

- Loop join point possible,
- Meaningful thanks to context exposure,
- Problem of loop selection would probably benefit from `pcflow`, `dflow` and even a possible `pdflow`.