**SPECULATIVE PARALLELIZATION OF pointer-based applications**

Adrian Tineo, Marcelo Cintra, and Diego R. Llanos

Dpt. Computer Architecture, University of Malaga
School of Informatics, University of Edinburgh
Dpt. Computer Science, University of Valladolid

**MOTIVATION:**
- Pointer-based applications pose a challenge for dependence detection in current parallelizing compilers
- Speculative parallelization is useful for running in parallel code sections that:
  - cannot be fully analyzed by the compiler, or
  - exhibit a small amount of dependencies

**KEY FEATURES:**
- We extend an all-software speculative parallelization engine based on sliding window to support pointer-based applications
- We create a scratch heap for each window slot, which provides a working space for threads
- 3 tables provide the needed support for scratch heap allocation, address translation and tracking possible conflicts between accesses

**OPTIMIZATIONS:**
- We can choose coarse- or fine-granularity of heap locations (example is for fine-granularity)
- The speculative heap can be reduced with help from compile-time analysis:
  - pointer analysis
  - def-use chains analysis
  - shape analysis

**Example program**

```c
// Pointer-based data structures
struct t1 { int data; struct t1 * nxt; };
struct t1 ** arr; // Data structure creation

// Speculatively parallelized loop
for(i=0; i<4; i++) {
    ptr = arr[i];
    val = ptr->data;
    ptr->data = val + 2;
    arr[i] = ptr->nxt;
}
```

**Speculative parallelization engine with support for pointer-based programs**

- At the start of the loop, a new scratch heap is created for every window slot, based on the current user heap, using the HAT.

- Each available thread takes the first free window slot and runs its block of statements.

- Scratch heaps are created empty. Values are fetched from the user heap as needed by load operations, using the HTT for translation.

- Heap accesses are recorded in the AST, with no need for memory fences. If a conflict arises, the offending thread is squashed.

- Non-speculative window slots that are done can be committed to the user heap, using the HTT and AST.

**Scratch heap 1**

**Scratch heap 2**

**Scratch heap 3**

**Scratch heap 4**

**User heap**

**Window slot 1**

**Window slot 2**

**Window slot 3**

**Window slot 4**

**HTT (Heap Translation Table):** translates the user heap addresses to the different scratch heaps

**HAT (Heap Allocation Table):** registers allocated pieces of memory in the heap, their size and type

**AST (Access State Table):** keeps track of accesses to scratch heaps. Possible states are: NotAcc, not accessed; ExpLd, read from user heap; Mod, modified; ExpLdMod, first read from user heap then modified