Support for Thread-Level Speculation into OpenMP

Sergio Aldea, Diego R. Llanos and Arturo Gonzalez-Escribano
Universidad de Valladolid, Spain
{sergio|diego|arturo}@infor.uva.es

8th International Workshop on OpenMP, IWOMP 2012, June 11-13, 2012, Rome, Italy.

Universidad de Valladolid

Grupo Trasgo
Universidad de Valladolid

Introduction

- **Manual development** of parallel versions of sequential applications is a **difficult task**. It requires:
  - In-depth knowledge of the problem.
  - Understanding of the underlying architecture.
  - Knowledge on the parallel programming model.
- **OpenMP** allows to parallelize code "avoiding" these requirements.
- Compilers' **automatic parallelization** only proceed when there is **no risk**.
- **Thread-Level Speculation (TLS)** can extract parallelism when a compile-time dependence analysis can not guarantee that the code is safely parallelizable.
- We have already developed a TLS runtime library.
- **Current goal**: To automatically transforms loops written in OpenMP syntax to benefit from speculative parallelization.

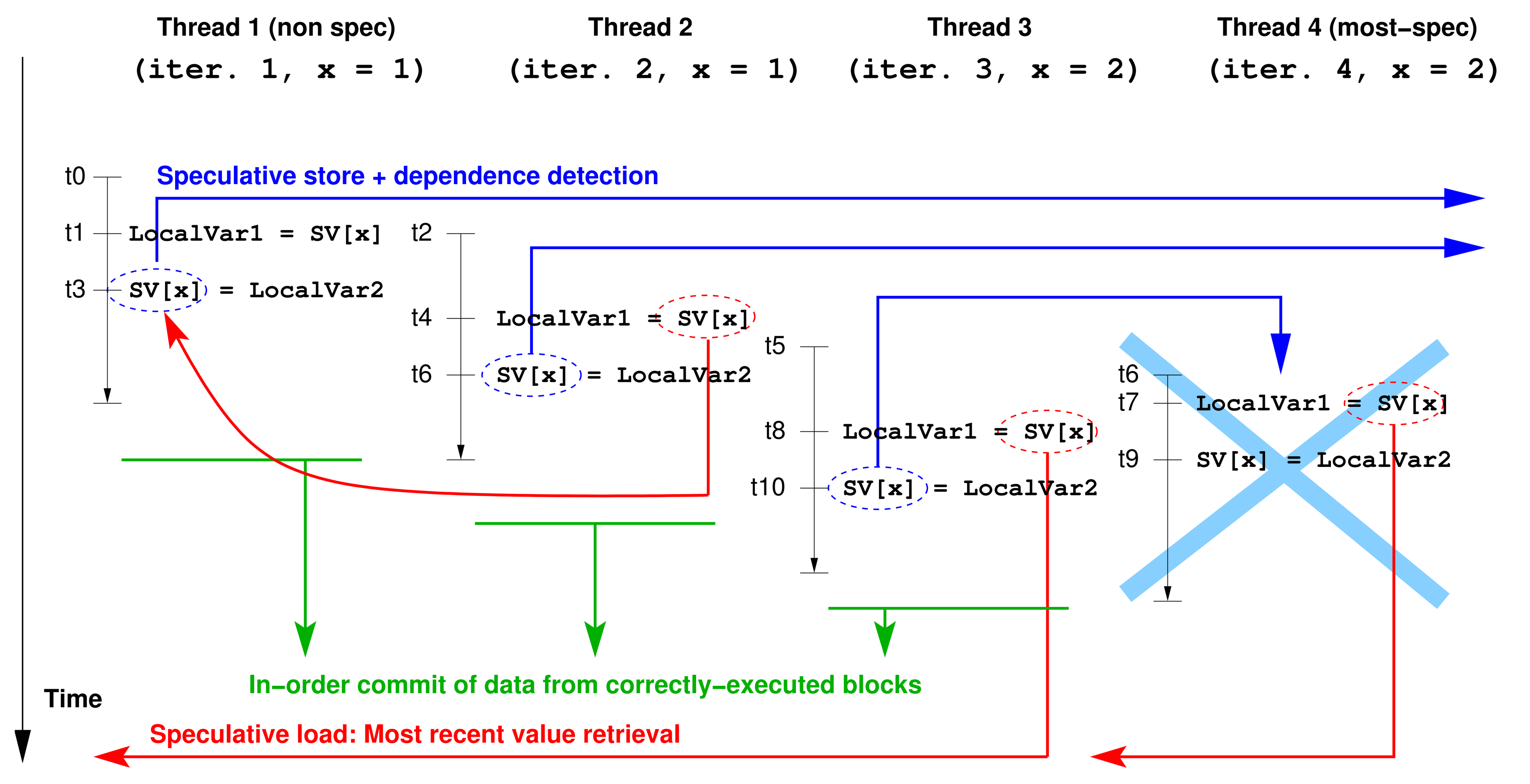
Our proposal

- **Goal:** Add TLS support into OpenMP.
- **New OpenMP clause**:

  #pragma omp parallel for \
  speculative(variable[,var_list])

- **Speculative variables** are those whose use can potentially lead to a dependence violation. They need to be monitored at run-time in order to obtain results.
- Programmer classifies variables in private, shared, and a new category: **speculative**.
- TLS should be **transparent** from the point of view of the programmer. If he/she is unsure about the use of a certain structure, he/she could simply label it as speculative. The compiler automatically will transform the code in order to speculatively parallelize the loop.

How Thread-Level Speculation works?

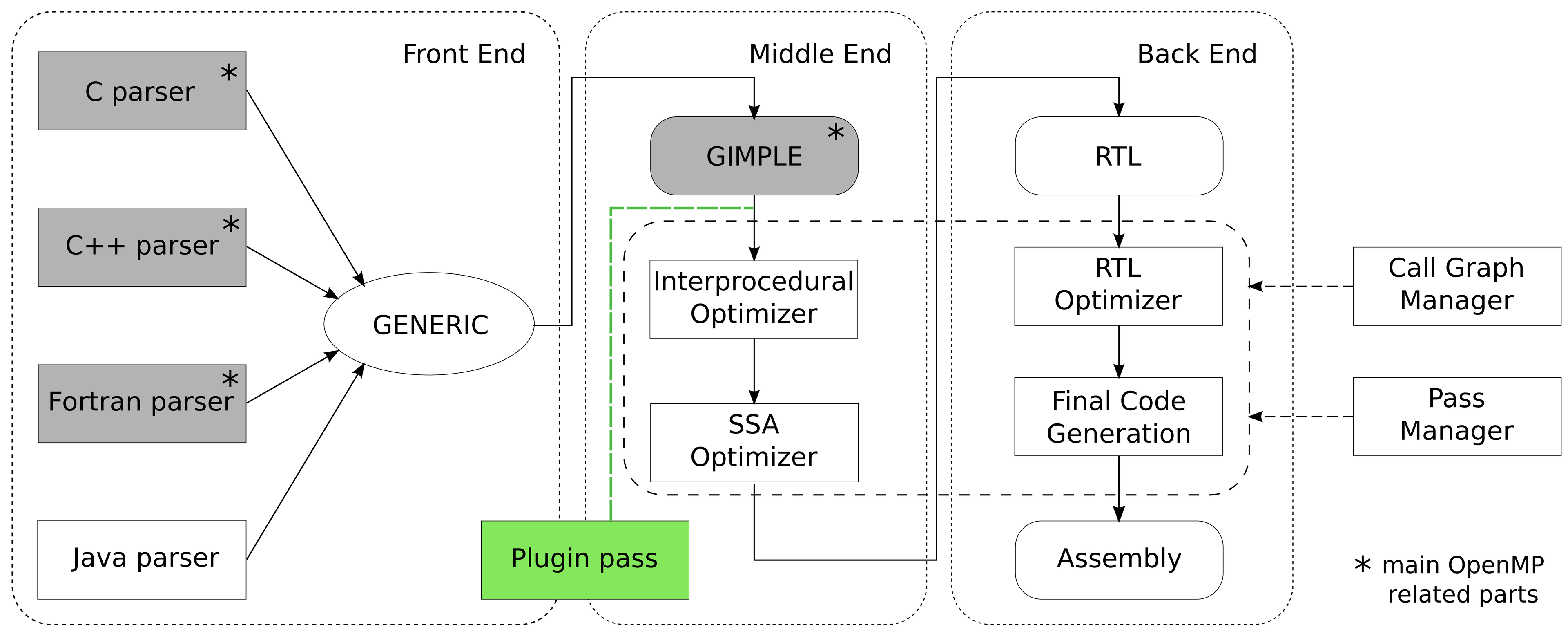


Our TLS is implemented using OpenMP for thread management.

Modifying GCC

- Use GCC as reference compiler.
- Since version 4.5, GCC can be extended by **plugins**:
  - Faster prototyping.
  - Easier modifications.
  - Extensibility: new compiler passes.
- The parser recognizes the new clause, and a new pass performs the transformations needed.
- Transformations are done before the compiler optimization passes.
- The new pass works with the **GIMPLE** representation.

GCC Architecture



Code Example

Original annotated

```
#pragma omp parallel for \
  private(a), shared (b) \
  speculative(v)

for (i=0; i < N; i++) {
  a = v[i];
  v[i] = b;
}
```

Code generated

```
specinit();
omp_set_num_threads(T);
specstart(N);
#pragma omp parallel for \
 private(a), shared (b) \
 private(engine_vars), shared(engine_vars) \
 shared(v)
{
  initSpecLoop(v, 1);
  specload(a, v, i);
  specstore(v, i, b);
  endSpecLoop(v, N);
}
```

- This transformations are done by the new pass **automatically**.
- It **detects** each reading from and writing into the speculative variable and **replaces** them for `specstore()` and `specload()` functions.
- It also add all the structures and functions needed to speculatively parallelize the code.

Conclusions

- Adding speculative support to OpenMP would greatly **increase the number of loops** that could be parallelized with this programming model.
- The programmer may label some of the variables involved as private or shared, using `speculative` for the rest.
- The parser detects the new `speculative` clause, and the new compiler pass **performs automatically all the transformations** needed to speculatively parallelize the loop.
- This process is **transparent to programmers**. They do not need to know anything about the speculative parallelizing model.
- Our proposal would let to transform *any* loop into a parallel loop.

Acknowledgements

This work has been partially supported by MICINN (Spain) and the European Union FEDER (CENIT OCEANLIDER, CAPAP-H3 network, TIN2010-12011-E, TIN2011-25639), and the HPC-EUROPA2 project (project number: 228398) with the support of the European Commission - Capacities Area - Research Infrastructures Initiative. Sergio Aldea is supported by a research grant of Junta de Castilla y León, Spain.