Due to the heterogeneity of parallel programming platforms today (MPI, openMP, Python's Threads and Multiprocessing modules), there is a need for a common approach which:

1. will free programmers from platforms' technical intricacies, and
2. will allow flexible computation, in which sequential and parallel executions produce identical deterministic results.

**Motivation**

**Objectives**

Our objective has been to develop EFL, a language which implements this common approach. With EFL we aim to implement safe and efficient parallel execution, in software, hardware, or both.

**Method**

EFL has been designed as a deterministic parallel programming language in order to guarantee safety of parallel executions. EFL blocks of code are embedded into a sequential host language program.

The sequential parts of the program are written in the host language.

EFL blocks:

- sequential code written in the host language ...
- EFL (if <a>b</a> { ...
- ... sequential code written in the host language ...
- EFL blocks:
  1. may be embedded into code written in any sequential host language.
  2. will be translated into parallel host language code.

An EFL pre-compiler has been implemented to do so.

Our first implementation is for Python.

**EFL Programming Model**

The EFL Programming Model imposes three kinds of restrictions in order to ensure deterministic parallelization:

- The programmer may call "pure" functions only.
- In and Out variables (but not InOut variables!).
- Once-only assignments.

**EFL Execution Model**

Flexible Computation is a basic concept of the EFL Execution Model:

A key aspect of Flexible Computation is well-defined flexible execution: parallel and/or sequential program execution orders of a program written according to the EFL Programming Model, will yield identical values.

**Conclusions and Further Work**

The work-span method was used to analyze EFL task trees which enabled us to estimate parallelism and speedup. Experiments were done to measure actual execution times and calculate real speedups on a specific platform.

Conclusions:

- An EFL pre-compiler (a parallel programming tool for creating safe and efficient parallelism) has been successfully implemented. Speedup increases were demonstrated by a case study comparing estimated and measured speedup for a family of task trees. Results show that speedup increases almost linearly with the size of a task tree.
- Several tracks of further development are in progress:
  
  a) The EFL pre-compiler is being developed for other host programming languages (C++, Java, C#, Fortran, etc.) and for other platforms (MPI, OpenMP, SCCOP, etc.)
  
  b) Serial Execution Mode, for debugging purposes.
  
  c) EFL curriculum, to teach how to implement serial and parallel algorithms using EFL.
  
  d) IDEs for EFL programming.
  
  e) Rewriting DEEPSAM with EFL and STM. DEEPSAM is a parallel evolutionary algorithm for protein structure prediction, written in Python upon the TINKER molecular modeling package.