Workpackage 4: Concrete Instantiation

Objectives

The objective of this workpackage is to show the validity of the conceptual model by providing a concrete example implementation of the skeleton library and to assess its efficiency on a representative benchmark suite (the outcome of WP7). This task is important as it should demonstrate: (1) how the programmer can use the skeletons in the conceptual model through a simple API from an imperative language enjoying wide acceptance among parallel programmers; (2) how benchmark programs can be concisely expressed through the API; (3) the efficiency of the library code on the benchmark compared with the levels of performance achieved by low level implementations using widespread parallel libraries (e.g. MPI, PVM...); (4) the portability of library code on different platforms and its ability to adapt to different targets. The task presents several challenges as concepts provided by skeletons like higher-order functions, partial application, currying, and polymorphism, are usually difficult to express in conventional imperative languages. Thus providing these concepts in an easy usable way to the conventional imperative programmer requires a careful engineering of the API and of its implementation. Moreover, the efficient implementation of a program built composing many skeletons requires in general the generation of a complex process net and non-trivial resource tuning. All this must be done on-the-fly before performing the actual computation required by the user. This requires fine tuning optimization algorithms to run fast and effectively using (possibly) incomplete information.

Description of Work

The activity required by this workpackage will consist of:

- selection of an imperative host language for the skeleton API and API definition
- implementation of the library functions in a portable way
- testing on the benchmark suite

Host language selection will require to assess the characteristics of different languages and devise innovative solutions to provide higher-order functions, polymorphism etc in a natural way to the conventional imperative programmer. We plan to build on the experience gained in the system Fact [**?**].

The implementation phase will need first to select a low level layer providing basic concurrency mechanisms. Probably the best candidate to do this is MPI, a well established message passing library which is available for a broad range of platforms and that is the state of the art for parallel programming. However, the usage of other emerging library and standards will be carefully evaluated. After fixing the low level, we will need to detail the implementation of the execution model. An important point to address here is how to reduce the complexity of the run time tuning of the global program structure. This tuning will need to address non-trivial problems such as obtaining smooth interaction among skeletons with possibly conflicting resource requirements and choosing different implementation strategies on the basis of the characteristics of the user-supplied parameters. All this should be done quickly and effectively to avoid adding too much overhead to the actual work to be performed for the user. Another major difficulty is the mapping of the task parallel skeletons, which have a non-SPMD flavor, to the SPMD style of programming provided by message passing libraries like MPI.

Finally, as soon as the prototype implementation is running, we will test benchmarks, including those developed in WP7, in order to check the efficiency of our library on different platforms. Based on the observations, we will fine-tune the implementation. The goal is that the efficiency reaches that of hand-coded largely used interfaces such as MPI.

Deliverables

We will develop:

- an API for the conceptual model
- a prototype implementation of the skeleton library,
- a report describing the design and implementation, including experimental results for the benchmarks on different hardware platforms.

We will make the system and its documentation freely available via the web.

Milestones and Expected Results

The choice of host language and definition of API will be finished after 3 months. 15 months will be needed for the design and implementation of the prototype. The report describing design, implementation, and run-time experiments will be finished after 21 months.