ANACLETO
a template based p3l compiler

S. Ciarpaglini, M. Danelutto, L. Folchi,

C. Manconi & S. Pelagatti

{ciarpagl,marcod,folchi,manconi,pelagatti}@di.unipi.it

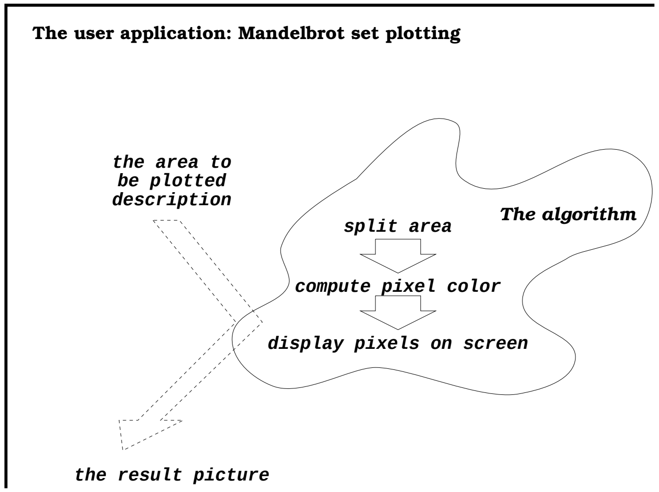
Project info at http://www.di.unipi.it/~susanna/p3l.html

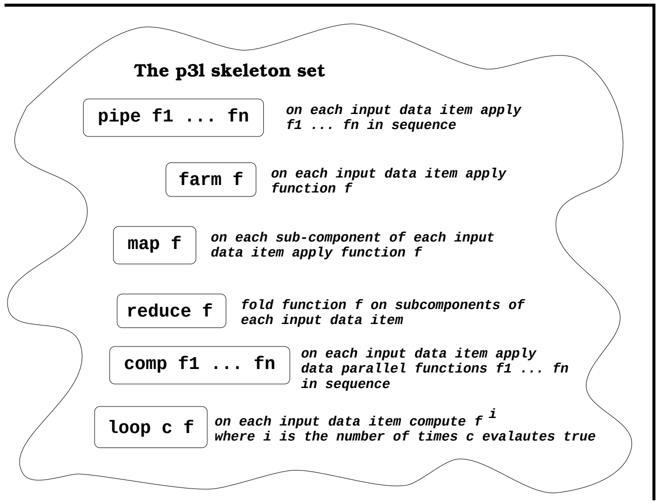
p3l

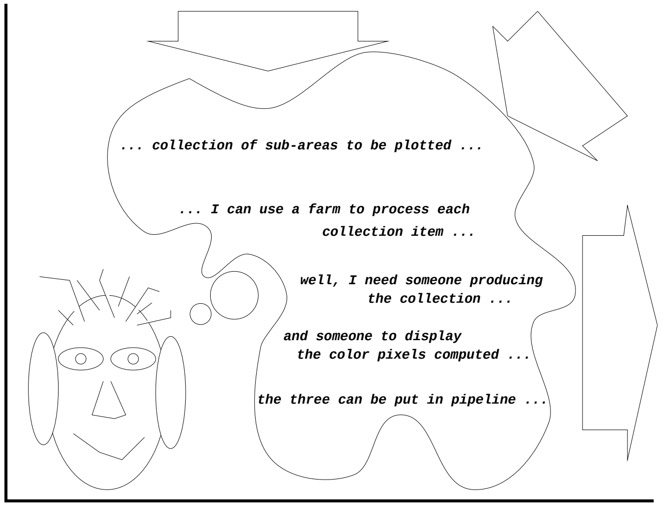
p3l is a *parallel coordination* language, designed in 1991, based on the *skeleton* programming model.
It belongs to the class of *restricted* computational models, in that the parallel structure of an application can only be expressed as a composition of the skeletons provided by the language.
Each skeleton models a well known pattern of parallel computation. There are *task parallel* skeletons, modeling pipelines and farms, *data parallel* skeletons, modeling usual data parallel computations such as *forall* (with or without data dependencies between the parallel activities) and *fold*, and *control* skeletons, modeling iterative computations and encapsulating sequential computations.
p3l is assumed to use a "host language" to express sequential computations as well as data structure declarations. Currently p3l uses ANSI C as the host sequential language.

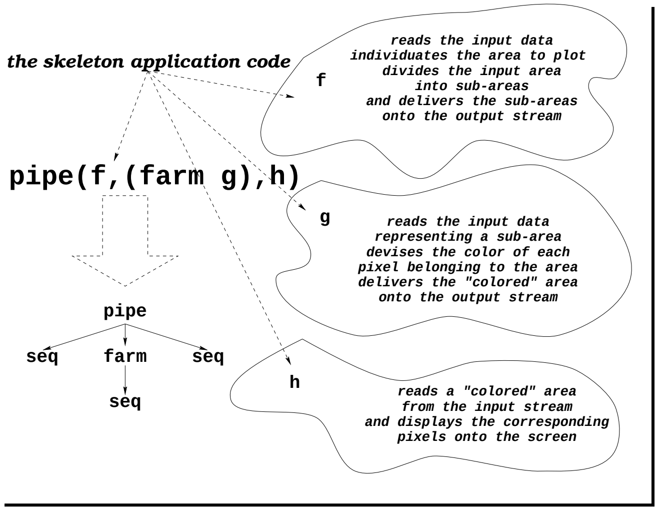
p3l applications are developed by users by modeling the behavior of the parallel algorithm they have in mind by means of a suitable nesting of the provided skeletons. Each skeleton has a number of parameters, modeling the computation and the input/output data specification, that have to be filled by the user in order to obtain a working program. The computation parameters may be filled by specifying either sequential code or other skeleton to be called.
Eventually, once all the skeleton structure and the sequential code have been specified, the p3l application code is to be compiled by a p3l compiler (such as ANACLETO) in order to get the target machine running code.

(have a look at `http://www.di.unipi.it/~susanna/p3l.html` for further info on the p3l project)









ANACLETO

ANACLETO is the first experimental p3l compiler generating MPI code and targeting (among the others) the Fujitsu AP1000 architecture.
The compiler exploits the template technology. A template represents an efficient way of implementing a skeleton onto a given parallel machine. It is basically built out of a *parametric process graph* and of some *analytical performance models*.
The process graph holds all the info needed to implement a process network that realizes the computation modeled by the corresponding skeleton onto the target architecture.
The performance models relate the expected performance of a template to parameters such as the amount of resources used in the template (number of processing elements used), target architecture potentialities (time spent in communicating data or in scheduling processes) and user code features (time spent in sequential computations).
Templates are stored into a *template library* that is consulted during the p3l compilation process. The library holds (possibly) multiple templates relative to the same skeleton, each one sporting different performance figures onto the target architecture.
ANACLETO first performs syntactical analysis on the source code and generates the skeleton tree. After that, each node in the tree is assigned a template. The template is chosen and instantiated in such a way that the overall application expected performance (measured using the analytical performance models stored in the library) is maximized.
Eventually, a collection of C files (with MPI calls) is produced along with a makefile.
Once compiled, those files produce an executable program that efficiently and correctly implements the p3l application onto the Fujitsu machine.

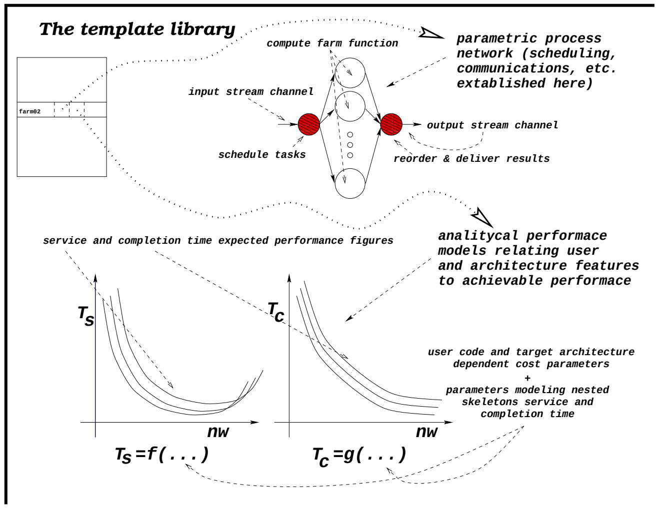
The whole compilation process realized by ANACLETO is completely automatic. The process includes profiling phases of user code (sequential) needed to properly evaluate the analytical performance models. The user is requested to supply significant sample input data in order to perform profiling.
In other words, once the user has prepared a source p3l file holding the application code, say `myapplication.p3l`, he/she has just to type:

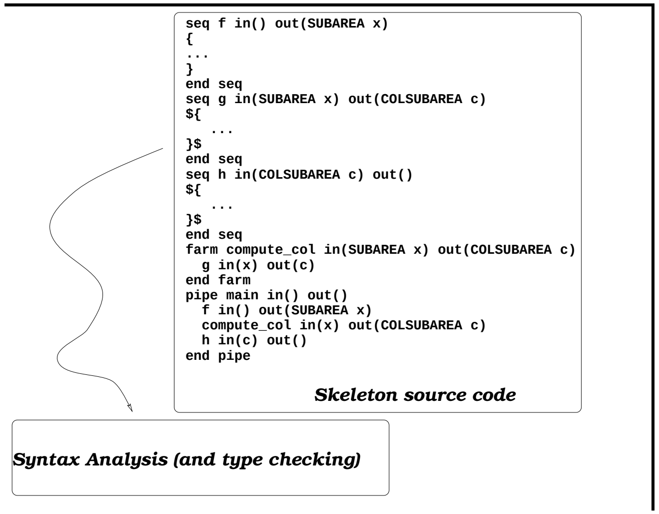
```
p3lcc -fuji -o myapplication myapplication.p3l
```

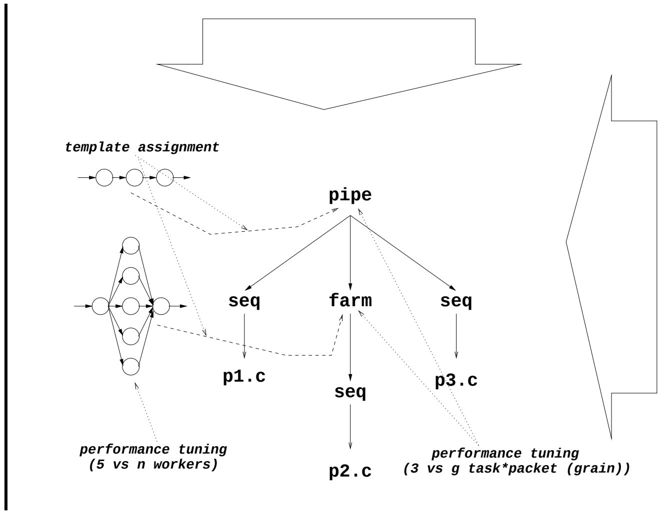
in order to get the runnable code, and

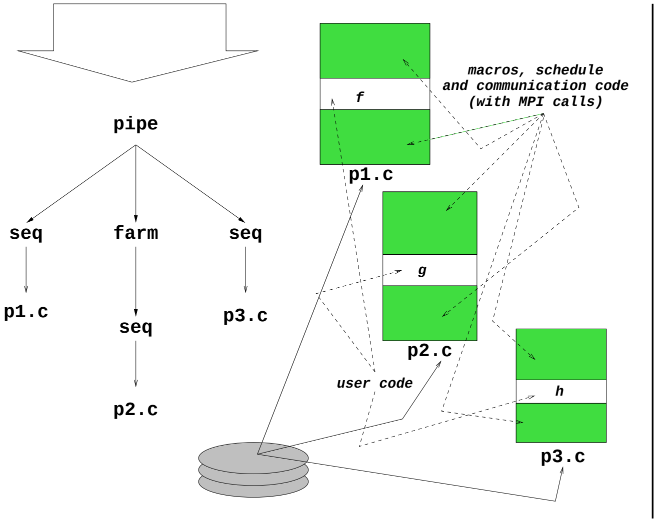
```
myapplication
```

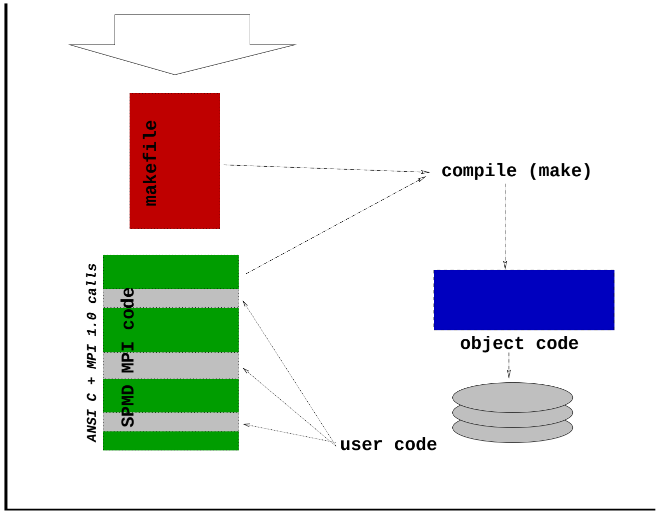
to run the application.

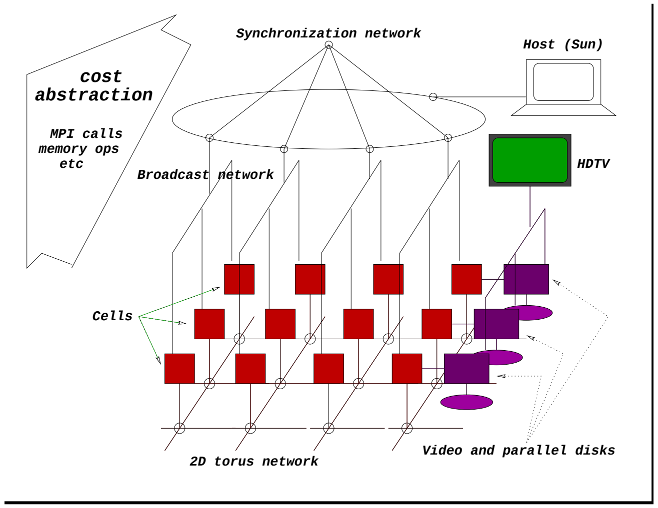






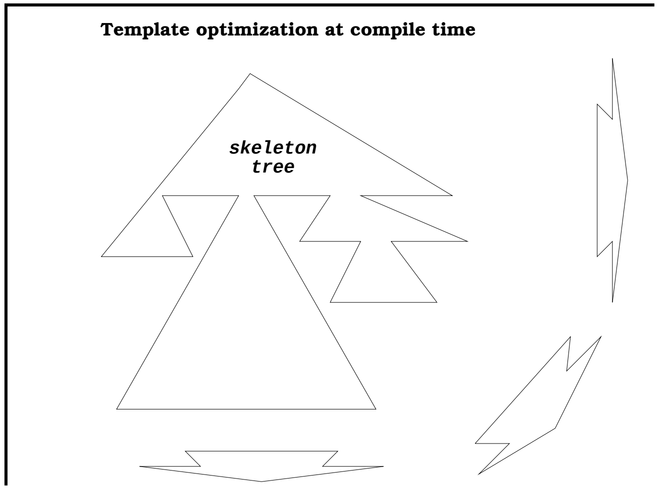


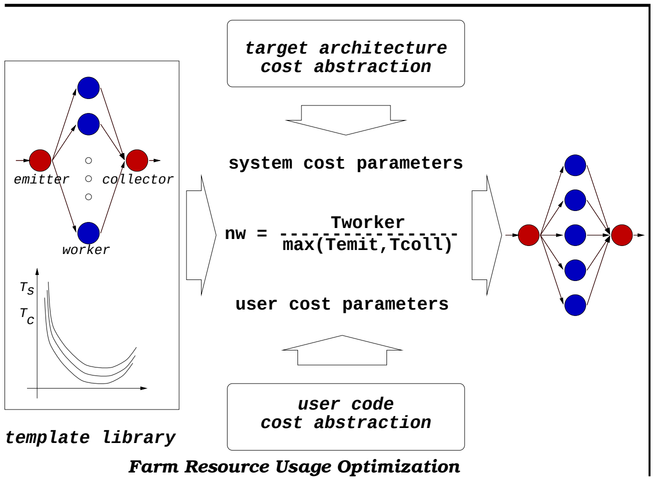




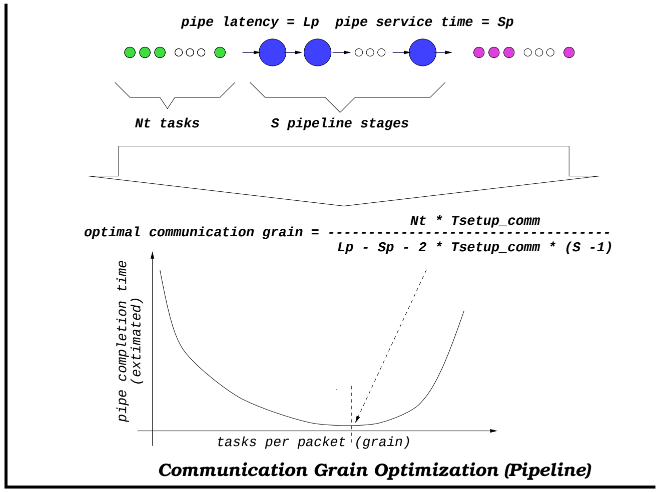
Optimization techniques

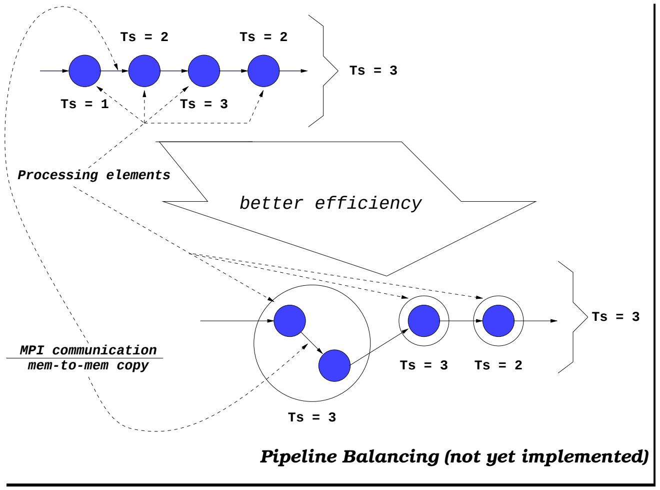
Whitin ANACLETO a number of "optimization" techniques are used, aimed at producing efficient code for the target architecture.
Such techniques (and heuristics) are applied while performing multiple visits of the skeleton tree building out the application parallel structure.
First, a template is assigned to each one of the skeletons appearing in the tree. Then the best "resource" parameters (e.g. the number of workers in a farm or in a map template) are determined. The computation of these parameters is performed consulting both the cost abstraction of user code and target architecture and the analytical performance models stored in the template library.
Then, the optimal communication grain is computed for each skeleton subtree, in such a way that the communication bandwidth of the target architecture is exploited.





Finally, redundant farm skeletons (i.e. the ones such that the computation of optimal number of workers leads to nw $\leq$ 2) are eliminated (exploiting the semantic equivalence `farm(f) = seq(f)`) and possibly the number of resources (processing elements) assigned to each template is reduced in order to match the machine resource count. The "reduction" process, taking away processing power from the implementation templates, is performed in such a way that the resulting template code preserves load balancing and efficiency in the processing power usage.
The final result of the optimization algorithms used within ANACLETO is an annotated skeleton tree. Annotations relative to each one of the skeletons in the tree record the template to be used to implement the skeleton along with all the parameters needed to instantiate the template. Such an annotated tree is eventually used to produce the ANSI C + MPI code that will be compiled in order to get the target architecture object code.

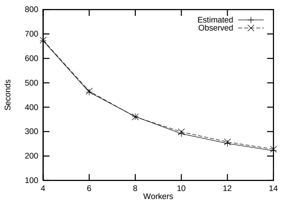




Assessments: analytical performance models

The main point in the development of ANACLETO has been the development of efficient (w.r.t. target machine) implementation templates for the skeletons supported by p3l.

In particular, we had to develop accurate analytical performance models in order to be able to embed in ANACLETO the knowledge necessary to compute the template parameters in such a way that good performance figures are obtained (i.e. no bottlenecks, no unuseful process, optimized communications, etc.) Experiments showed that the predicted performance measures actually match the observed ones with errors within 5%.

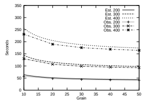


Estimated vs. observed completion time (farm with varying worker number)

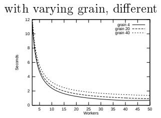
Assessments: communication grain optimization

Communications happening between the processing elements hosting processes of a template are processed in ANACLETO in such a way that the setup time of a communication is hidden as much as possible using standard double buffering techniques.
The impact of communication grain optimization (i.e. packing more data items within the same communication data packet) has been proved effective: with large data packets, varying the grain size may significantly improve the performance of the program.

Estimated vs. observed completion time



Pipeline with varying grain, different data sizes

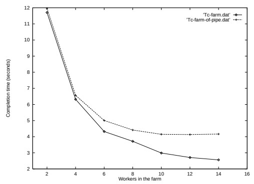


Farm with varying grain

Assessments: overall performance

p3l programs compiled by ANACLETO and run onto the Fujitsu AP1000 demonstrate real speedups.

Typical completion time figures show that performance increases with the number of processing elements used for execution, as far as the communication to computation time ratio is adequate.



In our experiments, if we force (by hand) different choices in the template parameter values computed by ANACLETO we obtain poorer performance figures.

Acknowledgments

We wish to thank all the people involved in the p3l project.

We also wish to thank the Imperial College in London for providing us the access to the Fujitsu AP1000 that enabled us to test the whole ANACLETO design and functionalities.