Automatic parallelisation of Python programs

Chris Lamb (cgl@dcs.warwick.ac.uk)

May 2008

The Python programming language is a high-level, multi-paradigm, object-oriented scripting language which is used in an ever-growing number of environments, both academic and industrial. Its popularity has been attributed in the most part to its design philosophy which emphasises programmer productivity and code readability, but also to a large number of high-quality libraries that have become available.

However, dispite a high level of interest, implementions of the language suffer from performance problems; depending on the benchmark used, Python programs can run up to 250 times slower the the corresponding programs written in C. This can be attributed to a number of reasons, including dynamic dispatch of all methods, 'boxed' arithmetic, extreme reflective capabilities and a general interpretative overhead.

A large number of solutions have been offered, but the overwhelming majority are not used (or are simply rendered obsolete) due to incompatibilities with the canonical implementation written in the C programming language ("CPython"). It is also clear that some tools are not used simply because they are *not* CPython, which poses interesting political questions for software tool uptake.

In conjunction with this, we have noticed a general trend towards 'multi-core' processors becoming *de-facto* in common consumer hardware, but whilst there are a growing number of tools for performing explicit parallelisation in Python, there is a lack of usable tools that can automatically (or even semi-automatically) parallelise Python programs that can exploit these additional processing units that are becoming commonplace.

My presentation will consist of a brief overview of the Python language, the difficulties associated with interpreting Python and CPython programs in parallel, and my attempts to develop a tool to automatically parallelise Python programs, which have concentrated on using the Cartesian Product Algorithm for localised type inference.