The StarPU Runtime System: Task Scheduling for Exploiting Heterogeneous Architectures

Olivier Aumage
Permanent junior researcher at LaBRI/INRIA (France)

Heterogeneous multi-core platforms, mixing regular cores and dedicated accelerators, are now so widespread to have become the nowadays canonical computing architecture. To fully tap into the potential of these hybrid platforms, both in terms of computation efficiency and power saving, pure offloading approaches in which the main core of the application runs on regular processors and offloads specific parts on accelerators, are not sufficient. The real challenge is to allow applications to fully use the cumulated computing power of the entire machine, by scheduling parallel jobs dynamically over the whole set of available processing units. The Inria Team RUNTIME has been studying this problem of scheduling tasks on heterogeneous multi/many-core architectures for many years, which led to the design of the StarPU runtime system. The StarPU runtime is capable of scheduling tasks over heterogeneous, accelerator-based machines. Its core engine integrates both a scheduling framework and a software virtual shared memory (DSM), working in close relationship. The scheduling framework maintains an up-to-date, self-tuned database of kernel performance models over the available computing units to guide the task/unit mapping algorithms. The DSM keeps track of data copies within accelerator embedded memories and features a data-prefetching engine, avoiding expensive redundant memory transfers and enabling task computations to overlap unavoidable memory transfers. Such facilities were successfully used in the field of parallel linear algebra algorithms, notably, where StarPU is now one of the target backends of the University of Tennessee at Knoxville's MAGMA linear algebra library. The StarPU environment typically makes it much easier to exploit heterogeneous multicore machines. Thanks to the Sequential Task Flow programming paradigm, programs may submit tasks to the StarPU engine using the same logical order as the sequential version, thus preserving initial algorithm layouts and loop patterns.