Study of GPU code generation techniques, including auto-parallelization, auto-tuning and skeleton-based code generation
Cedric Nugteren
22-02-2011

Background:
Multi-core and many-core architectures are emerging and becoming a commodity. Many experts believe heterogeneous processor platforms including both GPUs and CPUs will be the future trend.

Languages such as CUDA are specifically designed for general purpose GPU programming. Even so, the mapping process of most applications remains non-trivial. Furthermore, there is only a tiny fraction of programmers able to achieve an optimal hardware usage for their applications. Achieving this is non-trivial, having to deal with mappings of distributed memories, caches, and register files. Additionally, the programmer is exposed to parallel computing problems such as data-dependencies, race-conditions, synchronization barriers and atomic operations.

In order to use heterogeneous platforms efficiently, GPU programming has to change. Now, only a fraction of programmers are able to benefit from the GPU's computing power, while in the future many more programmers need to have access to these resources. Different techniques exist to generate high-performance code, some using annotations, some introducing new languages or frameworks, and some using auto-parallelization or auto-tuning techniques.

Code generation techniques:
Example code generators for GPUs are:

- Lee, S.; Grover, V.; Chakravarty, M. & Keller, G. GPU Kernels as Data-Parallel Array Computations in Haskell EPHAM 09': Exploiting Parallelism using GPUs and other Hardware-Assisted Methods, 2009
- Svensson, J.; Claessen, K. & Sheeran, M. GPGPU kernel implementation and refinement using Obsidian Procedia Computer Science, 2010, 1, 2059 - 2068

These code generators use source-to-source compilers, auto-tuning, pragma-based parallelization, skeletons, templates, and so on. Some introduce new tools, new languages or use a functional programming language. In all the above mentioned papers, comparison between their competitors is not made often, since this is not trivial.
**Tasks:**
Your task is to compare different code-generation techniques for GPUs and to create a comprehensive study of all techniques. In this way, an overview will be made of all advantages/disadvantages of the current techniques. To achieve this, your tasks could include:

- Literature study of the existing techniques (e.g. auto-tuning, skeletons)
- Selection of a suitable benchmark set
- Literature study of the different GPU code generators
- Code generation for your benchmark set with these existing code generators. This will lead to results in terms of performance and usability (among others).
- Evaluation of the advantages/disadvantages of the techniques in general (e.g. auto-tuning, skeletons, pragmas) and the specific GPU code generators (hiCUDA, pyCUDA, PGI Accelerator, CUDA-lite, SkePU, etc.)
- A proposal for a new GPU code generator based on the most suitable technique and the advantages of the best code generators.

**Importance:**
Existing papers on GPU code generation do not compare with others. This is because: (1) it is not trivial to set everything up and to make a fair comparison, and (2) the algorithms tested in the papers work well with their own code generator, but they might not support others. Without an evaluation, it is difficult to quantify the quality of these code generators.