21. PROGRAMMING LANGUAGES AND SOFTWARE TECHNOLOGY

JUDIT NYÉKYNÉ GAIZLER, Associate Professor; ISTVÁN VETŐ, Associate Professor; GERGELY FELDHOFFER, Assistant Professor

PROGRAMMING LANGUAGES

The importance of reliability of the programs has come to the centre of interest with the development of large systems. The aim is to write "correct" programs which exactly perform their tasks, as defined by the requirements of the specification.

A theory, i.e. a methodology which supports the programmer perform his practical work to be well-considered is very important. Abstraction - as a technique of thinking - is substantial in the education of professional programmers of the future. In the same time, we have to agree with Bertrand Meyer that the “language independent methodology” is “as useful as a bird without wings”, and: “In software perhaps more than anywhere else, thought is inseparable from its expression. To obtain good software, a good notation is not sufficient; but it is certainly necessary.” That’s the reason why we try to balance in our teaching already at BSc level between methodology and tools – programming languages, environments, etc.

Algorithm and software correctness are often discussed and educated separately. We look for ways to blend concepts as weakest precondition into the use of debuggers and static analyzers. Usual solutions like precondition assertion by logic-error exceptions and its analysis by debugger are part of our MSc education program.

GPU SUPPORT IN COMPILERS

There are quite a number of research and development projects on GPU programming in these years since the raw computational power of GPUs is in general higher than CPU by a magnitude. Most of the publications in this topic used to describe the application of a given algorithm to GPU architecture. The problem of algorithm application to a given architecture contains multiple layers of abstraction. The higher level of abstraction used to describe the parallelization scheme, and the lower level is the actual GPU code which determines the memory mapping (coalescing), the use of shared memory, CPU-GPU transfer, etc. Solutions of lower level abstraction tasks are often commonly known as best practice on the given problem. This situation is similar to low level CPU programming techniques, which are today highly automated in optimizers used by compilers. Our concept is to introduce a compiler which is capable to generate GPU enabled code from standard high level programming language.

A GPU enabled compiler should generate GPU code, such as OpenCL, which is a high level programming language. It should take care of memory transfer and representation. We succeed to create a proof-of-concept compiler plugin which is capable of these features, and is integrated into GCC. The higher level of abstraction is encoded here as a lambda function which is a C++ feature in the latest standard. The programmer has to use a custom template function library to access the GPU features in this version. Each template function represents a high level parallelization scheme, such as for_each() or cumulate(). The supported features inside the lambda function include (non-recursive) function call, operator overloading, and

many more. Examples of not supported features are the recursion, polymorphism and pointer aliasing based tricks.

SOFTWARE TECHNOLOGY

The category of Software Technology includes different disciplines, related to each other, but having its own specialties. The staff of PPCU Faculty of IT carries on numerous theoretical works on the fields of sciences concerned by the following subjects, included in the education program at our Faculty:

Basics of software technology: requirements engineering, software development processes and methodologies, project planning, basic questions of software quality and standards.

Software design and evaluation: methods and tools for software design and implementation, OO analysis and design, modeling tools and practices, agile software development methodology (Extreme Programming, SCRUM, etc.), software reengineering, configuration management, as well as evaluation of software systems.

Integration of information systems: disciplines of integrated information systems, models and architectures, classes and handling of heterogeneity, the goals of integration, systems integration strategies, types of integration, middleware and integration standards, integration architectures, like SOA.

Design patterns: Types and role of the classical OO design patterns in Java and C++, architectural patterns in software development.

Our activity is extended to study the architecture, structure and functions of complex data processing application systems, like ERP, CMS and different types of eBusiness software. We are also dealing with the standards and tools of Business Process Modeling.

PUBLICATIONS

[1] A. Rák, G. Feldhoffer, G. B. Soós, and Gy. Cserey. Standard C++ Compiling to GPU with Lambda Functions, In Proceedings of 2010 International Symposium on Nonlinear Theory and its Applications (NOLTA 2010), Krakow, Poland, 2010