The rise of the GPGPU-compilers

Painting "High Rise" made by Huma Mulji

If you read The Disasters of Visual Designer Tools you'll find a common thought about easy programming: many programmers don't learn the hard-to-grasp backgrounds any more, but twiddle around and click a program together. In the early days of BASIC, you could add Assembly-code to speed up calculations; you only needed tot understand registers, cache and other details of the CPU. The people who did that and learnt about hardware, can actually be considered better programmers than the C++ programmer who completely relies on the compiler's intelligence. So never be happy if the control is taken out of your hands, because it only speeds up the easy parts. An important side-note is that recompiling easy readable code with a future compiler might give faster code than your optimised well-engineered code; it's a careful trade-off.

Okay, let's be honest: OpenCL is not easy fun. It is more a kind of readable Assembly than click-and-play programming. But, oh boy, you learn a lot from it! You learn architectures, capabilities of GPUs, special purpose processors and much more. As blogged before, OpenCL probably is going to be the hidden power for non-CPUs wrapped in something like OpenMP.
Evolution
The first stage of compiler-development for GPUs would be some special language to a shader-language like OpenGL. Since awareness of limits gets known in time, the optimisations would eventually pass by the shader-language. Also the special language would get easier to program in time. I.e. in CUDA 3 you see that in a few years the language got faster in many aspects, and also at the driver-side there have been many changes. The second step (what's happening now) is automatic recognition of possible to-be-accelerated parts of code. We know that OpenMP-enriched programs can be rewritten to run (partly) on the GPU so now only the compiler has to be told in which cases and how. OpenCL - being an open standard - is perfect for an intermediate language, also for debugging purposes; an intermediate language we need, since for each new hardware the optimisations are different. Such a study has been done using CUDA: "OpenMP to GPGPU: A Compiler Framework for Automatic Translation and Optimization" [PDF, by Seyong Lee, Seung-Jai Min and Rudolf Eigenmann.

I want to stress that understanding hardware architectures stays important for GPGPU and any other programming language with performance in mind. In 2010 and 2011 you'll still see OpenCL in the light and before you know it's all hidden in libraries and handled by the compiler; so learn! The only difference is that the programmer still must understand where the software is to be used on.

Current GPGPU compilers

The reason I wanted to take a week time to find all GPGPU-compilers is that I got some 404-pages and projects with no changes in a few months time. Where did those projects go? Actually the techniques are so much dependent on driver-optimisations, that most projects are abandoned or now target CUDA or OpenCL instead of a shader-language - I did not expect this to be already happening or more gradually, even if I wrote it above. There are more and more wrappers and compilers which target OpenCL or CUDA, for

which a separate "wrappers and libraries"-post will be written for. So we have the following projects left:

Nvidia CUDA: Currently the most grown-up GPGPU-compiler, started in 2007. It uses a LLVM-based compiler.]
Khronos Group OpenCL: The industry standard for GPGPU working in a growing number of processors (AMD, Nvidia, ARM, IBM Cell). LLVM, GCC and other compilers can be used.]
PathScale ENZO: In collaboration with CAPS (with their HMPP-compiler) they want to put an alternative in the market. Sold for $2495,-]
CodePlay Offload: only targets IBM's Cell-processor.]
Sh: When RapidMind was acquired by Intel, the code for targetting GPUs was opensource(d).]
Microsoft DirectCompute: It actually is mostly on top of CUDA and OpenCL, but since it's nicely integrated into DirectX I can't leave it out.]

If you're missing a project, please let me know.

Disclaimer: PrimeCortex/StreamHPC gives trainings in GPGPU/OpenCL. We like to show you how to learn GPGPU and OpenCL and like to share our enthusiasm via this blog, but of course our training is more thorough.