Automatic and Portable Mapping of Shared Memory Programs to OpenCL for GPU-based Heterogeneous Systems
Dominik Grewe (dominik.grewe@ed.ac.uk)

Overall Scheme
Realizing the potential of GPU-based heterogeneous systems is challenging due to the complexity of programming. We have developed a compiler-based approach to automatically generate optimised OpenCL code from shared memory OpenMP programs. A key feature of our scheme is that it leverages existing transformations, especially data transformations, to improve performance on GPU architectures. As not all programs are suitable for GPU execution it uses predictive modeling to automatically determine if it is worthwhile running the OpenCL code on the GPU or OpenMP code on the multi-core host.

Our compiler first identifies kernels within the OpenMP program. Several code transformations are applied before the OpenCL kernels are generated. This code is passed to our feature extraction tool to collect program features. At runtime an offline-built Machine Learning (ML) model is evaluated on these features to decide whether it is beneficial to run the OpenCL code on the GPU. If not, the original OpenMP code is executed on the multi-core host.

Code Optimisations
GPUs can only achieve peak performance if data is accessed efficiently. While shared memory programs are generally optimised for intra-thread locality, GPU code should be optimized for inter-thread locality, where successive threads access successive memory locations, to enable memory coalescing. We therefore apply several transformations to improve memory behaviour on the GPU:

- Loop interchange
- Global index reordering
- Dynamic index reordering

Intra-thread vs inter-thread locality (row-major)

- Single thread accesses block of memory.
- Interleaved access.

Dynamic index reordering
When no global data layout is optimal, data can be rearranged dynamically for parts of the program. This is only beneficial if the performance gains of improved memory accesses outweigh the cost of reordering.

Predicting the Mapping
To predict whether GPU execution is beneficial or not we build a predictive model using machine learning (ML) techniques. For a set of training programs we find the optimal mapping and use this information together with the programs’ features to build the ML model.

ML model (decision tree)

Results
Speedup over single-core execution for all NAS benchmarks averaged across input sizes. Using only a multi-core CPU or a GPU leads to average speedups of 2.80x and 1.71x on System #1 (left) and of 2.95x and 0.63x on System #2 (right). Our model achieves an accuracy of 92% and 97% for predicting the right device. This translates into an average speedup of 4.51x and 4.20x respectively.

While some programs do not benefit from GPU execution (eg. lu) others achieve significant performance improvements (eg. ep). Selecting the right device is thus crucial for performance on heterogeneous systems.

Case Study: The sp benchmark
Performance of the sp benchmark for different input sizes and after applying different transformations:
L=loop interchange, DD=dynamic data transformations.
The naive OpenCL code shows poor performance. Loop interchange improves memory access patterns and thus improves performance. Further applying dynamic data transformations boosts the performance of the OpenCL code significantly. However, only for larger inputs is the GPU execution able to outperform CPU execution. Mapping techniques must therefore be sensitive to changes in both input size and code optimisations.