Performance Estimation Framework for Automated Exploration of CPU-Accelerator Architectures

Tobias Kenter kenter@upb.de

1. Challenge
Our goal is the design space exploration and evaluation of CPU-Accelerator architectures. For each set of hardware parameters, the speedup depends on an appropriate HW/SW partitioning for each benchmark. Since this step is usually performed manually, classical (co-)simulation works only for a limited design space.

2. Approach
We present a fully automated high level toolflow based on the LLVM infrastructure with a focus on speed and flexibility. We perform application characterization with profiling, code analysis and cache simulation. Based on this data, a combined performance estimation and HW/SW partitioning shows the potential of different hardware parameters.

3. Parameters to investigate
- Performance of CPU and accelerator
- Clock speeds and parallel execution z(CPU), z(ACC)
- Size / resources of accelerator
- Interface latencies $\lambda_e, \lambda_i$
- Cache hierarchy
- Private / shared caches, Local memory
- Cache sizes
- Cache latencies $\lambda_m$

4. Application characterization
Profiling:
LLVM profiling features deliver:
- Execution count of basic blocks $n(B_j)$ and instructions
- Execution count of edges between Blocks $n(B_i, B_j)$

Code analysis:
Analysis of LLVM intermediate code combined with execution count of edges deliver:
- Register dependencies between basic blocks
-Pull or push method

Memory / cache simulation:
Memory instrumentation and simulation of a cache hierarchy deliver:
-Cache level serving a memory access
-Write back through shared cache
-Data analysis allows reuse of results

5. Example code

6. Example mapping to architecture

7. Greedy partitioning approach
Block level partitioning:
- Individually move each basic block to accelerator if size fits
- Compute possible speedup
- Select basic block with best speedup/size ratio
- Repeat until no more speedup possible

Multi level partitioning:
- Basic blocks are still investigated as partitioning objects
- Additional partitioning objects: Loops, Functions

8. Performance model
Total program runtime:
$$t = t_c + t_m + t_e$$

Execution time:
$$t_c = \sum_{k: I_k = op} n(I_k) \cdot \epsilon(p(I_k))$$

Memory access time:
$$t_m = \sum_{k: I_k = Id} \times \sum_{j=1}^{n(I_k)} \lambda_m(v(I_k^j))$$

Control transfer time:
$$t_e = \sum_{(l,m)} n(B_l, B_m) \cdot \lambda_c$$

Register value transfer time:
$$t_e = \sum_{R} \min(n(B_l) \cdot \lambda_{r.push}, n(B_l) \cdot \lambda_{r.pull})$$

9. Results
a) Overview: several benchmarks need multi level partitioning for reasonable results
b) Sample mapping of ADPCM: loop needs to be partitioned in one step, block level partitioning doesn't work
c) Varying the parameter interface latency: multi level partitioning much more robust for higher latencies