ARTIFACT DESCRIPTION APPENDIX for the SC18 poster submission

"Applying the Execution-Cache-Memory Performance Model: Current State of Practice"

A. Abstract

The ECM (Execution-Cache-Memory) model is an analytic, resource-based performance model for steady-state loop code running on multicore processors. Starting from a machine model, which describes the interaction between the code and the hardware, and static code analysis it allows an accurate prediction of the runtime of sequential loop code. Together with a scaling assumption, it also gives a performance scaling prediction. This poster summarizes the current state of practice in constructing and applying the ECM model, points out problems and open questions, and applies the model to three nontrivial use cases.

B. Description

B.1. Check-list

- Program code
    - The benchmark source for the "Solar" and "CG" examples: https://blogs.fau.de/hager/files/2018/08/AD-ECM-Solar-BB.zip
    - The github repository for the "Blue Brain" example can be found at https://github.com/sharkovsky/barebones_C_miniapp_CoreNeuron_kernels
- Compilation: The Intel compiler in version 18.0 Update 2 was used for all benchmarks. Performance-relevant compiler options were "-O3 -xHOST -fno-alias -qopenmp" except for the Blue Brain kernels where "-O2" was used.

  The README and Makefile for Solar and CG show how to build the binaries.

- Data set: For the "Solar" code, the problem was set up as a cubic lattice. The sizes can be read from the graphs on the poster. The "Blue Brain" package contains appropriate problem size setups. For the "CG" example the size was set to 1000 x 40000 grid points (i.e., the matrix had 4 x 10^7 rows).
- Run-time environment: Ubuntu 16.0.4 LTS, transparent huge pages activated, NUMA balancing activated, SMT activated (but only physical cores used in benchmarks), Cluster on Die for Haswell activated (7 cores per NUMA domain). Clock speed was always fixed to the base clock frequency (including the Uncore) using the likwid-setFrequencies tool (version 4.3.2). This was 2.2 GHz on the Ivy Bridge and 2.3 GHz on the Haswell.
- Hardware: As given on the poster. Memory bandwidth limits were obtained using the likwid-bench tool (version 4.3.2) using

  ```
  likwid-bench -t copy_avx -w M0:500MB:7:1:2
  likwid-bench -t load_avx -w M0:500MB:7:1:2
  ```

  on Haswell. Substitute 7 with 10 on Ivy Bridge.

- Experiment workflow: Statistical variations were small enough to be neglected in all measurements.

- Publicly available: The Blue Brain miniapps are publicly available. The source for the CG and Solar apps will be made available via the AD link on the poster.

B.2. How software can be obtained (if available)

See links above

B.3. Hardware dependencies

The performance results depend on the particular CPU models. For other CPUs, the modeling process must be re-done.

B.4. Software dependencies

LIKWID 4.3.2, Intel compiler 18.0 Update 2

B.5. Datasets

The only required input for all benchmarks is the problem size (see above).

C. Installation

No installation procedures required (apart from the SW dependencies above). Binaries were started right from the build directories.

D. Experiment workflow

Experiments consisted in running the binaries with appropriate problem sizes. Experiments were repeated to rule out significant statistical variations.

E. Evaluation and expected result

Since the poster is about performance modeling, comparing modeling predictions with measured performance/runtime is the first step of validation. Additional validation via data traffic measurements (optional LIKWID marker API instrumentation) was done for the Solar and Blue Brain cases.