Performance Modeling and Engineering Using Kerncraft

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Friedrich-Alexander-University of Erlangen-Nuremberg, Regional Computing Center Erlangen
Julian Hammer <julian.hammer@fau.de>, Georg Hager (advisor), Gerhard Wellein (advisor)
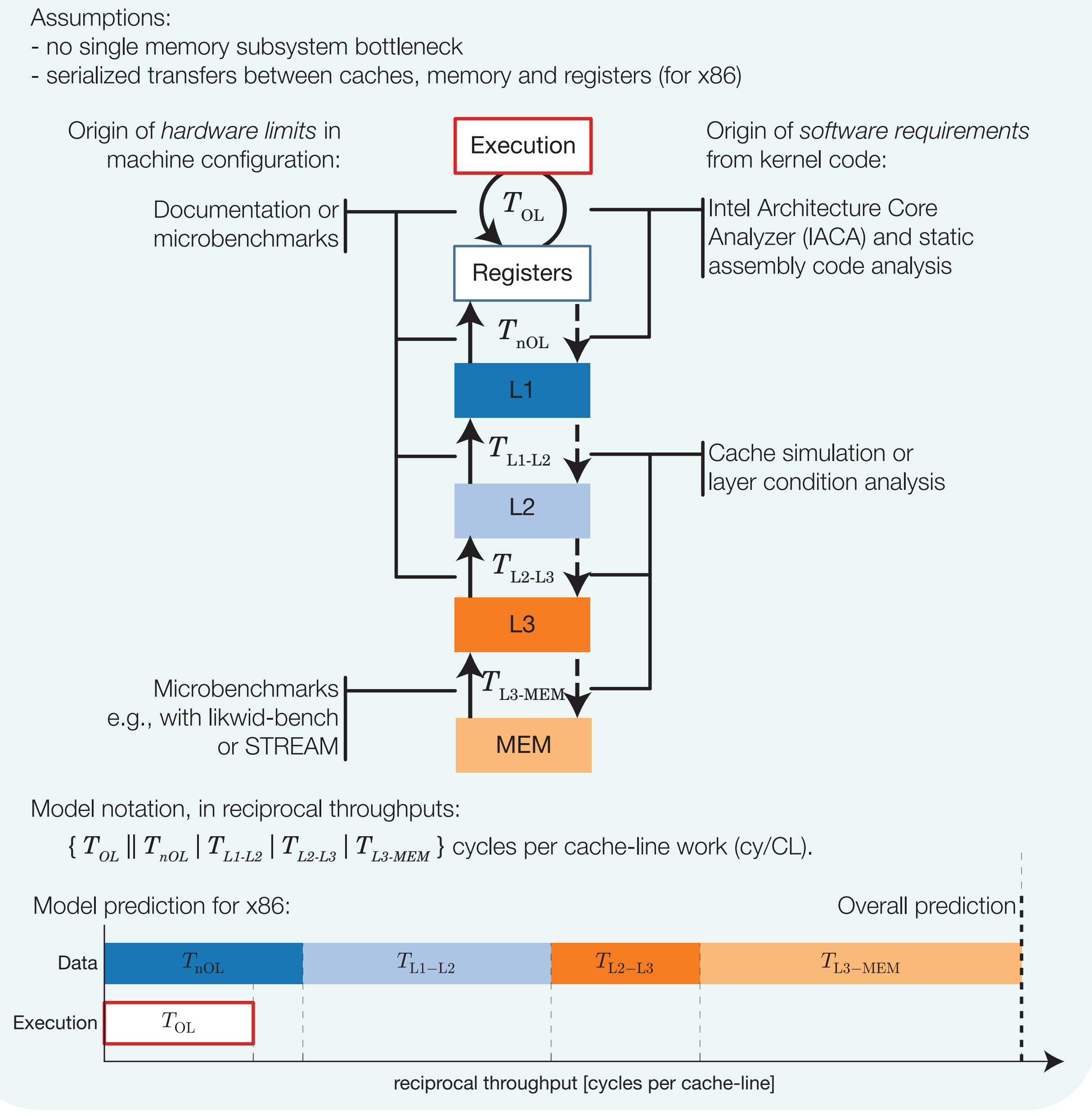
RRZE HPC

Goal

Predict performace, increase cache utilization and select the most efficient core configuration for regular algorithm implementations (streaming, stencil) on current multi-core CPUs.

Approach

1. Build Execution-Cache-Memory model
2. Predict serial performance and optimal scaling
3. Analytically select optimal spatial blocking factors

Kerncraft[1]

User Input

Kernel Code

Standard compliant C99 code, with no additional branches in loops.

3D-long-range kernel code[2], as used for following examples:

```
double U[M][N][N], V[M][N][N], ROC[M][N][N];
double c0, c1, c2, c3, c4, lap;

for(int k=4; k < M-4; k++) { for(int j=4; j < N-4; j++) { for(int i=4; i < N-4; i++) {
    lap = c0 * V[k][j][i]
        + c1 * (V[ k ][ j ][i+1] + V[ k ][ j ][i-1] + V[ k ][j+1][ i ] + V[ k ][j-1][ i ]
        + c1 * (V[ k ][j-1][ i ] + V[k+1][ j ][ i ] + V[k-1][ j ][ i ] )
        + c2 * (V[ k ][ j ][i+2] + V[ k ][ j ][i-2] + V[ k ][j+2][ i ] )
        + c2 * (V[ k ][j-2][ i ] + V[k+2][ j ][ i ] + V[k-2][ j ][ i ] )
        + c3 * (V[ k ][ j ][i+3] + V[ k ][ j ][i-3] + V[ k ][j+3][ i ] )
        + c3 * (V[ k ][j-3][ i ] + V[k+3][ j ][ i ] + V[k-3][ j ][ i ] )
        + c4 * (V[ k ][ j ][i+4] + V[ k ][ j ][i-4] + V[ k ][j+4][ i ] )
        + c4 * (V[ k ][j-4][ i ] + V[k+4][ j ][ i ] + V[k-4][ j ][ i ] );
    U[k][j][i] = 2.f * V[k][j][i] - U[k][j][i] + ROC[k][j][i] * lap;
}}}
```

Accesses to array V (stencil)

Machine Configuration

Semi-automatically gathered using LIKWID tools[4], contains information and benchmarks on the micro- and macroarchitecture of the compute node.

Sandy-Bridge Machine configuration file, as used for following examples:

CPU & compiler information

```
model name: Intel(R) Xeon(R) CPU E5-2680 0 @ 2.70GHz
sockets: 2
cores per socket: 2
```

Memory subsystem

```
memory hierarchy:
    - level: L1
      cache per group: {'sets': 64, 'ways': 8, 'cl_size': 64, # 32 kB
                        'replacement_policy': 'LRU', 'write_allocate': True,
                        'write_back': True, 'load_from': 'L2', 'store_to': 'L2'}
      cores per group: 1
      cycles per cacheline transfer: 2
```

Benchmark results

```
benchmarks: {measurements: {MEM: {results:
    update: [18.91 GB/s, 32.43 GB/s, 37.28 GB/s, 39.98 GB/s, 40.99 GB/s, 40.92 GB/s, 40.61 GB/s, 40.34 GB/s]}}}
```

Execution-Cache-Memory (ECM) Model[0] for x86

Assumptions:
- no single memory subsystem bottleneck
- serialized transfers between caches, memory and registers (for x86)

Origin of *hardware limits* in machine configuration:
- Documentation or microbenchmarks
- Cache simulation or layer condition analysis
- Microbenchmarks e.g., with likwid-bench or STREAM

Origin of *software requirements* from kernel code:
- Intel Architecture Core Analyzer (IACA) and static assembly code analysis

Model notation, in reciprocal throughputs:

$$\{\ T_{OL}\ \|\ T_{nOL}\ |\ T_{L1\text{-}L2}\ |\ T_{L2\text{-}L3}\ |\ T_{L3\text{-}MEM}\ \} \text{ cycles per cache-line work (cy/CL)}.$$

Model prediction for x86:

reciprocal throughput [cycles per cache-line]

ECM[0] & Roofline[3] Model

Automatic Modeling

For both ECM and Roofline, we need to predict the amount of data loaded and stored in each memory level, as well as execution time for all arithmetic instructions.

8 iterations / 1 cache-line of work: 232 FLOPs (88 MULs, 136 ADDs, 8 SUBs)

	First Addr.	Last Addr.	First Addr.	Last Addr.	First Addr.	Last Addr.	First Addr.	Last Addr.
STOREs	U[k][j][i]	U[k][j][i+7]						
LOADs	V[k-1][j][i]	V[k-1][j][i+7]	V[k][j-1][i]	V[k][j-1][i+7]	V[k+1][j][i]	V[k+1][j][i+7]	V[k][j+1][i]	V[k][j+1][i+7]
	V[k-2][j][i]	V[k-2][j][i+7]	V[k][j-2][i]	V[k][j-2][i+7]	V[k+2][j][i]	V[k+2][j][i+7]	V[k][j+2][i]	V[k][j+2][i+7]
	V[k-3][j][i]	V[k-3][j][i+7]	V[k][j-3][i]	V[k][j-3][i+7]	V[k+3][j][i]	V[k+3][j][i+7]	V[k][j+3][i]	V[k][j+3][i+7]
	V[k-4][j][i]	V[k-4][j][i+7]	V[k][j-4][i]	V[k][j-4][i+7]	V[k+4][j][i]	V[k+4][j][i+7]	V[k][j+4][i]	V[k][j+4][i+7]
	V[k][j][i-4]	V[k][j][i-3]	V[k][j][i]	V[k][j][i+7]	V[k][j][i+8]	V[k][j][i+11]		

In-Core with IACA

Intel Architecture Core Analyzer (IACA) allows detailed analysis of instruction scheduling and execution. It predicts throughput and critical path latency of inner-most loops. We employ it in the following process:

Kernel Code → Compilable Code → Assembly → IACA Marked Assembly → Binary → IACA Analysis → Scaled IACA Analysis — Unrolling Factor

Cache with pycachesim

Released under AGPLv3
github.com/RRZE-HPC/pycachesim

Cache simulation allows us to predict a wide variety of cache behaviors and architectures using your cach simulation library, pycachesim.

Simulated Cache Hierarchy

- Accessed locations and lengths
- Statistics on HITs, MISSs and STOREs
- Tracking of occupancy and mapping

Initialization is key to predicting steady-state behavior. By simulating enough accesses until the largest cache is completely occupied and execute exactly one cache-line worth of accesses, the resulting cache state change provides insight into the costs of one CL work-load.

Layer Conditions[5] Model

Layer conditions (LCs) give an analytical approach to select optimal blocking factors for LRU-based cache hierarchies. With a 3D kernel, here the 3D long-range stencil, three different layer-conditions can be found:

1D layer condition | 2D layer condition | 3D layer condition

- stream head (uncached)
- cached element

To relate LC dimensions to accesses, we consider slices which contain a subsets of accesses which lie in the same dimension. E.g., all elements in the same plane are part of the same slice for 2D layer conditions. When computing relative offsets between accesses, only offsets within slices are considered. Offsets accross slice boundaries are handled by the second term in paranthesis.

Under the assumption of an LRU cache, we get the following equation:

$$C_{req.} = \left(\sum L_{rel.offsets} + \max(L_{rel.offsets}) * n_{slices} \right) * s$$

- Required size
- Sum over all relative offsets between accesses
- Longest relative offset over all slices
- Number of slices in dimension
- Bytes per element

Example: 2D LC for 3D long-range stencil. By adding up the number of stream heads in the 2D LC illustration and arrays (u and ROC) in the code, we get:

$$n_{slices} = 9 + 1 + 1$$

The longest relative offsets are from V[k][j+1][i] to V[k][j+2][i], thus:

$$\max(L_{rel.offsets}) = N$$

Summing up all relative offsets, yields:

$$\sum L_{rel.offsets} = 6N + 8 * 1 + 2(N - 4) = 8N$$

Cache size requirement for 2D LC:

$$C_{req.} = (8N + N * 11) * 8 \tfrac{\text{Bytes}}{\text{element}} = 152N \text{ Bytes}$$

Interactive LC Calculator:
https://rrze-hpc.github.io/layer-condition/#calculator

Benchmark Mode
Excluded for brevity

Future Work

In-core simulation — Development of IACA has not been updated recently and does not support Intel Broadwell. We therefore need a replacement, which will also enable support of other architectures.

LLVM-Polly integration — Kerncraft provides the information to guide compile-time decisions, if used in combination with a compiler. We are currently evaluating this in LLVM-Polly.

Graph applications — Going from regular to irregular is a challenging task, which we are currently researching with performance modeling of graph algorithms. If feasable this will become a target of future Kerncraft releases.

References

[0] Jan Treibig and Georg Hager. Introducing a performance model for bandwidth-limited loop kernels. In Parallel Processing and Applied Mathematics, pages 615–624. Springer Science + Business Media, 2010. doi: 10.1007/978-3-642-14390-8_64.

[1] Julian Hammer, Georg Hager, Jan Eitzinger and Gerhard Wellein. Automatic loop kernel analysis and performance modeling with Kerncraft. In PMBS '15 Proceedings of the 6th International Workshop on Performance Modeling, Benchmarking, and Simulation of High Performance Computing Systems. doi: 10.1145/2832087.2832092.

[2] Tareq Malas, Georg Hager, Hatem Ltaief, Holger Stengel, Gerhard Wellein, and David Keyes. Multicore-optimized wavefront diamond blocking for optimizing stencil updates. SIAM Journal on Scientific Computing, 37(4):C439–C464, 10 2015. doi: 10.1137/140991133.

[3] S. Williams, A. Waterman, and D. Patterson. Roofline: An insightful visual performance model for multicore architectures. Commun. ACM, 52(4):65–76, 2009. doi: 10.1145/1498765.1498785.

[4] J. Treibig, G. Hager, and G. Wellein. LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments. Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures, San Diego CA, September 13, 2010. doi: 10.1109/ICPPW.2010.38.

[5] Gabriel Rivera and Chau-Wen Tseng. Tiling optimizations for 3D scientific computations. In Proceedings of the 2000 ACM/IEEE conference on Supercomputing (SC '00). IEEE. doi: 10.1109/sc.2000.10015.

Results and Validation

ECM Prediction

```
$ kerncraft -p ECM -m E5-2680_2.7.yaml \
    3d-long-range.c -D N 1024 -D M 1024
================== kerncraft ==================
3d-long-range-stencil.c -m E5-2680_2.7.yaml
-D M 1000 -D N 1000
===================== ECM =====================
{ 56.6 || 53.0 | 40.0 | 24.0 | 50.3 } cy/CL
{ 56.65 \ 93.0 \ 117.0 \ 167.3 } cy/CL
[...]
$
```

Here we predict the reciprocal throughput for one work-unit (cache-line length) of an array with inner dimensions of 1000x1000 elements. Due to numerous elements required, loading is the limiting factor, but there is no single significant bottleneck showing:

Parameter Study

```
$ kerncraft -p ECM -p LC -p Benchmark \
    -m E5-2680_2.7.yaml 3d-long-range.c\
    -D N 10-5000:100:log10 -D M [...]
[... analysis results for 100 data points ...]
$
```

Analytical modeling (Roofline and ECM) and measurement sweep over the inner dimension, with data residing in main memory:

Optimal Scaling Point

```
$ kerncraft -p ECM -m E5-2680_2.7.yaml \
    3d-long-range.c \
    -D N 1000 -D N 1000
================== kerncraft ==================
3d-long-range-stencil.c -m E5-2680_2.7.yaml
-D M 1000 -D N 1000
===================== ECM =====================
[...]
saturating at 4 cores
$
```

If we assume perfect scaling of all resources except for the main memory bandwidth, we expect the performace to max out at four cores.

Blocking Suggestion

```
$ kerncraft -p LC -m E5-2680_2.7.yaml \
    3d-long-range.c -D N 1024 -D M 1024
================== kerncraft ==================
3d-long-range-stencil.c -m E5-2680_2.7.yaml
-D M 1024 -D N 1024
====================== LC =====================
[...]
2D Layer-Condition:        3D Layer-Condition:
L1: N = 216                L1: N = 19
L2: N = 1725               L2: N = 55
L3: N = 137971             L3: N = 488
```

By blocking the code according to the predicted layer conditions, performance does not decrease with increasing N: