Performance Engineering by Means of Automated Performance Modeling

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

Vision: Automatically predicit single-node loop kernel performance on current and future architectures.

Kerncraft[1]

Released under AGPLv3 on github.com/RRZE-HPC/kerncraft

Kernel

Req. computational and data resources

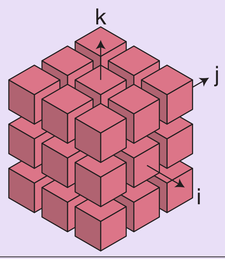
Machine

Offered computational and data resources

Restrictions:
> no branches in inner loop body
> only affine loops and array indices
typically for dens linear algebra, stencil, streaming codes

C Code (stable)

```
double a[M][N][N], b[M][N][N], s;
for(int k=1; k<M-1; ++k) for(int j=1; j<N-1; ++j) for(int i=1; i<N-1; ++i)
  b[k][j][i] = ((a[k][j][i] + a[k][j][i-1] + a[k][j][i+1]) +
    (a[k][j-1][i] + a[k][j-1][i-1] + a[k-1][j][i-1]) + (a[k][j+1][i] +
    a[k][j+1][i-1] + a[k][j+1][i+1]) + (a[k-1][j][i] + a[k][j-1][i+1] +
    a[k-1][j][i+1]) + (a[k+1][j][i] + a[k+1][j][i-1] + a[k+1][j][i+1]) +
    (a[k-1][j-1][i] + a[k-1][j-1][i-1] + a[k-1][j-1][i+1]) +
    (a[k-1][j+1][i] + a[k-1][j+1][i-1] + a[k-1][j+1][i+1]) +
    (a[k+1][j-1][i] + a[k+1][j-1][i-1] + a[k+1][j-1][i+1]) +
    (a[k-1][j+1][i+1] + a[k+1][j+1][i] + a[k-1][j+1][i-1])) * s;
```



OR

LLVM IR/Polly (in-development)

LLVM-Polly finds suitable loop nests and exports them

OR

Metadata (prototype)

High-level description of loop nest resources

Semi-Automatic Gathering (prototype)

YAML based description of hardware architecture

Ivy-Bridge Machine configuration file:

CPU & compiler
```
model name: Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz
compiler: icc: -O3 -xAVX -fno-alias
```

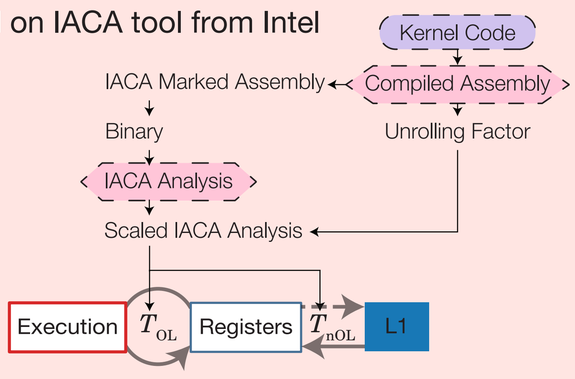
Memory subsystem
```
memory hierarchy:
    - level: L1
      cache per group:
        {'sets': 64, 'ways': 8, 'cl_size': 64, # 32 kB
         'replacement_policy': 'LRU', 'write_allocate': True,
         'write_back': True, 'load_from': 'L2', 'store_to': 'L2'}
      cycles per cacheline transfer: 2
```

Benchmark results
```
benchmarks: {measurements: {MEM: {results:
    update: [18.91 GB/s, 32.43 GB/s, 37.28 GB/s, 39.98 GB/s, ...]}}
```

Generated automatically, with additional input provided in vendor documentation.

Intel Architecture Code Analyzer[4] (stable)

Based on IACA tool from Intel



Very accurate, but closed source, only supports Intel CPUs and not an officialy supported product.

OR

Open Source Architecture Code Analyzer (in-development)

Open source replacement for Intel's IACA

1. Gathering of relevant instructions from typical codes.
2. Measurement of latency, throughput and overlap of instructions and loads with microbenchmarks.
3. Generation of the in-core model.
4. Application to given loop-nest assembly.

In-core prediction

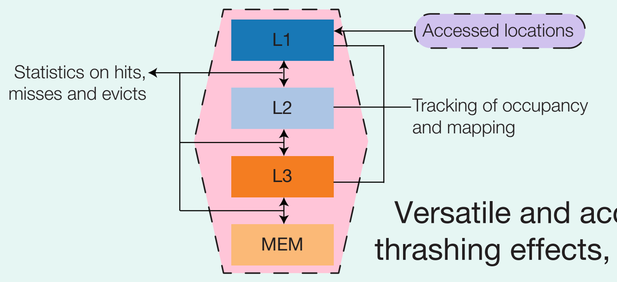
Computation and loading from/storing to L1

Cache prediction

Hits/Misses in all cache levels

Cache Simulator (stable)

Complete cache simulation using pycachesim



Versatile and accurate for thrashing effects, but slow.

OR

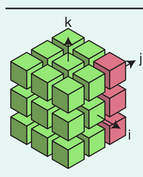
Layer Conditions[3] (stable)

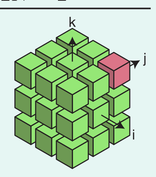
Analytical model for inclusive, least-recently-used caches

$$C_{\text{req}} = \sum(L_{\leq t}) + t * \text{count}(L_{>t})$$

$$L = \{ \underbrace{\infty}_{\text{first access to a}}, \underbrace{\infty}_{\text{first access to b}}, \underbrace{N^2-2N-2, N^2-2N-2,}_{\text{e.g., \&a[k][j+1][i+1] - \&a[k][j-1][i-1]}}$$
$$\underbrace{N-2, N-2, N-2, N-2, N-2, N-2,}_{\text{e.g., \&a[k][j-1][i+1] - \&a[k][j-1][i-1]}} \underbrace{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}_{\text{e.g., \&a[k][j][i+1] - \&a[k][j][i]}}\}$$

$t = N-2$	C_{req}	$t = N^2-2N-2$
$N \leq 410$	$\leq 32\,\text{kB}$	$N \leq 32$
$N \leq 3277$	$\leq 256\,\text{kB}$	$N \leq 91$
$N \leq 327680$	$\leq 25\,\text{MB}$	$N \leq 905$
18	Hits	26
10	Misses	2





Fast and may predict optimal tiling, restricted to stencil-like patterns and steady cache behavior. Extension to dens linear algebra is in the works.

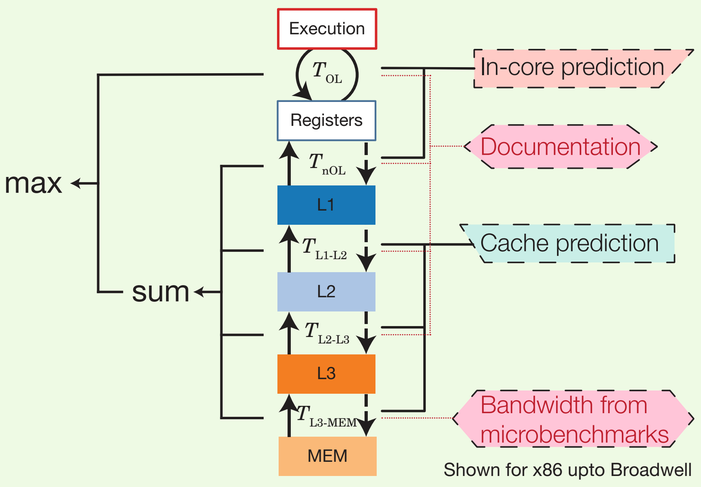
Interactive LC Calculator:

Performance Model

Compilation of runtime prediction

Execution-Cache-Memory[0] (stable)

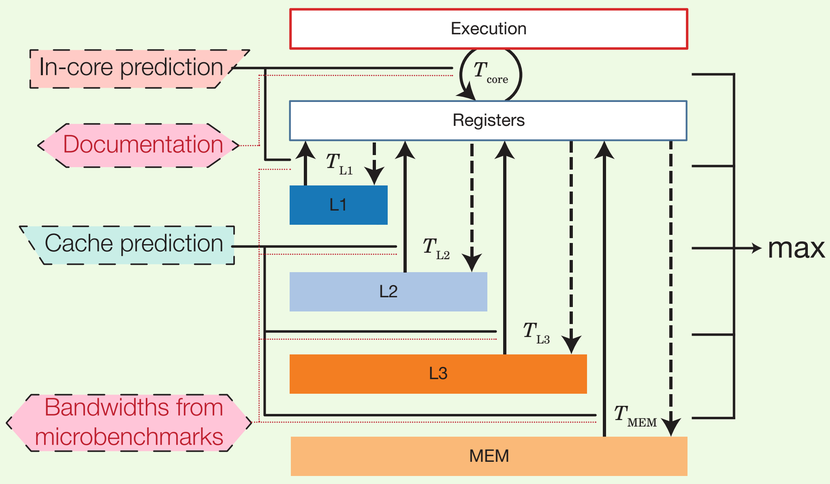
Roofline based model with serial assumption on data path and cache bandwidths based on documentation



In-socket scaling is derived by increasing the number of cores, until a non-scaling resource (e.g., memory bandwith) maxes out.

Roofline[2] (stable)

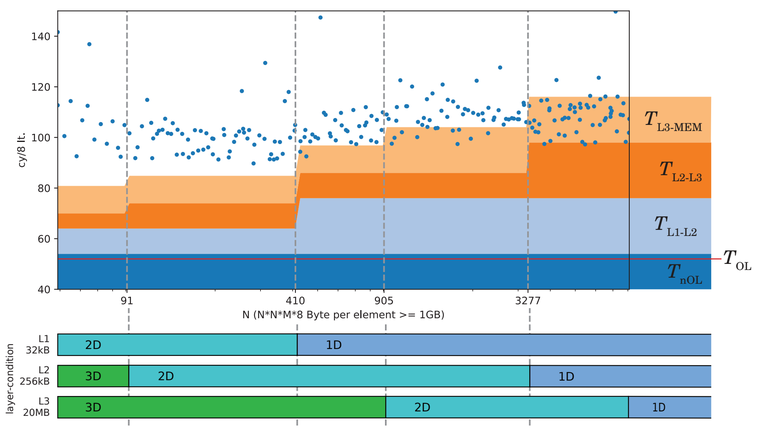
Well known performance model with overlap assumption on all bottlenecks, based on measured bandwidths



The Roofline model is less accurate if multiple data-transfer bottlenecks have almost the same performance and do not overlap perfectly.

Model Predictions and Conclusions

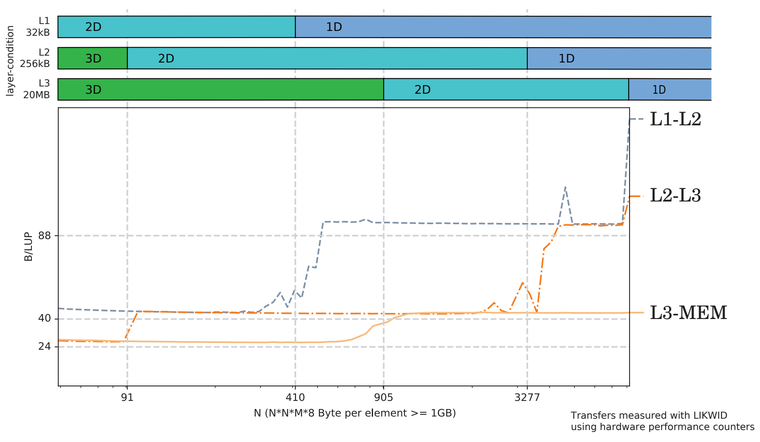
ECM model prediction and measurement



Key components are relevant to:

> compilers
tile size selection
> architecture researchers
impact of architectural changes on runtime
> offline optimizations
preevaluation of instruction mix
> energy-efficient computing
selection of cores and frequencies
> software developers
explain and predict behavior

Data transfer prediction and measurement



FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Supported by
Federal Ministry of Education and Research
SKAMPY, METACCA & SeaSite
DAAD Deutscher Akademischer Austauschdienst German Academic Exchange Service
FITweltweit

[0] J. Treibig and G. 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] J. Hammer, G. Hager, J. Eitzinger and G.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] 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.

[3] J. Hammer, J. Eitzinger, G. Hager and G. Wellein. Kerncraft: A Tool for Analytic Performance Modeling of Loop Kernels. Tools for High Performance Computing 2016, pp 1-22, 2016. doi: 10.1007/978-3-319-56702-0_1.

[4] I. Hirsh and Gideon S., Intel Corp. Intel Architecture Code Analyzer. Version 2.2, March 13, 2017. https://software.intel.com/en-us/articles/intel-architecture-code-analyzer