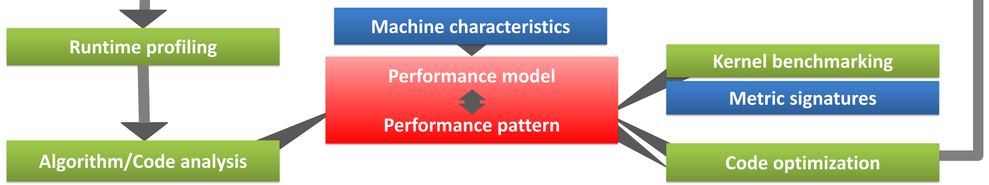
Pattern-Driven Node-Level Performance Engineering

Jan Treibig, Georg Hager, and Gerhard Wellein
University of Erlangen-Nuremberg, Erlangen Regional Computing Center (RRZE), 91058 Erlangen, Germany

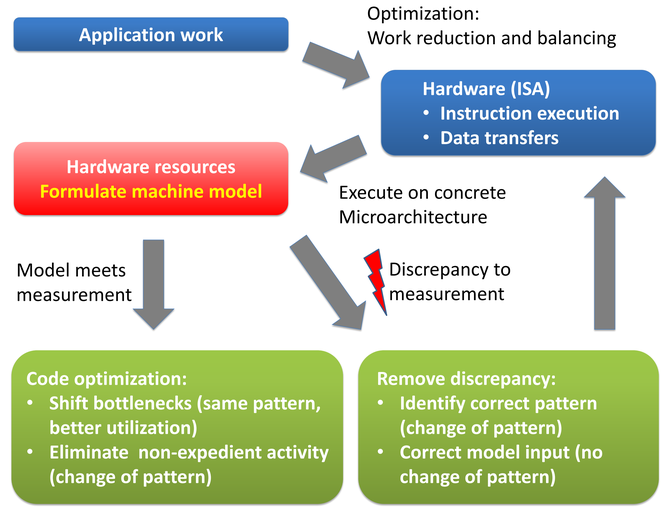
	Pattern		Performance behavior	Metric signature & LIKWID [7] performance group(s)
Hazards	Bandwidth saturation		Saturating speedup across cores sharing a data path	Bandwidth meets BW of suitable streaming benchmark (MEM, L3)
	ALU saturation		Throughput at design limit(s)	Good (low) CPI, integral ratio of cycles to specific instruction count(s) (FLOPS_*, DATA, CPI)
	Inefficient data access	Excess data volume	Simple bandwidth performance model much too optimistic	Low BW utilization / Low cache hit ratio, frequent CL evicts or replacements (CACHE, DATA, MEM)
		Latency-bound access		
	Micro-architectural anomalies		Large discrepancy from simple performance model based on LD/ST and arithmetic throughput	Relevant events are very hardware-specific, e.g., memory aliasing stalls, conflict misses, unaligned LD/ST, requeue events
	False sharing of cache lines		Large discrepancy from performance model in parallel case, bad scalability	Frequent (remote) CL evicts (CACHE)
	Bad ccNUMA page placement		Bad or no scaling across NUMA domains, performance improves with interleaved page placement	Unbalanced bandwidth on memory interfaces / High remote traffic (MEM)
	Pipelining issues		In-core throughput far from design limit, performance insensitive to data set size	(Large) integral ratio of cycles to specific instruction count(s), bad (high) CPI (FLOPS_*, DATA, CPI)
	Control flow issues		See above	High branch rate and branch miss ratio (BRANCH)
Work related	Load imbalance / serial fraction		Saturating/sub-linear speedup	Different amount of "work" on the cores (FLOPS_*); note that instruction count is not reliable!
	Synchronization overhead		Speedup going down as more cores are added / No speedup with small problem sizes / Cores busy but low FP performance	Large non-FP instruction count (growing with number of cores used) / Low CPI (FLOPS_*, CPI)
	Instruction overhead		Low application performance, good scaling across cores, performance insensitive to problem size	Low CPI near theoretical limit / Large non-FP instruction count (constant vs. number of cores) (FLOPS_*, DATA, CPI)
	Code composition	Expensive instructions	Similar to instruction overhead	Many cycles per instruction (CPI) if the problem is large-latency arithmetic
		Ineffective instructions		Scalar instructions dominating in data-parallel loops (FLOPS_*, CPI)

The Performance Engineering development cycle

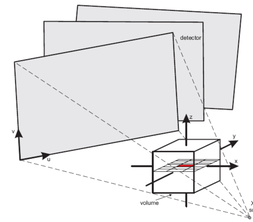


We propose a systematic performance engineering process constructed around a diagnostic performance model. The performance model is a tool forcing the developer to acquire knowledge about the interaction of their code with a specific system architecture. In the present work the performance modeling approach is combined with the notion of *performance patterns* [1]. Patterns give the developer a concrete plan how to proceed to understand the observed performance, how to formulate a suitable model, and to decide on optimizations. Patterns condense common knowledge and experience in an accessible way. They help in understanding what the limiting factors of a code are, which quantitative model is suitable to describe the performance, and finally which optimization techniques are available to improve the performance.

Basic Motivation

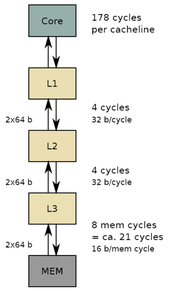


Example: Computed tomography volume reconstruction [3]



RabbitCT benchmark:
http://www.rabbitct.com

Backprojection



Work reduction optimization

LIKWID-perfctr analysis

Load imbalance pattern

Select optimal OpenMP schedule

Roofline analysis

Memory-bound algorithm!

Memory BW saturation pattern

ECM Model analysis using IACA

ALU saturation, Pipelining issues, Code composition patterns

Replace divide with pipelined reciprocal

Apply SIMD vectorization

Use SMT capabilities

Discrepancy between model and measurement

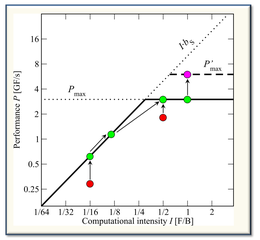
Result of effort:

5-6 x improvement against initial parallel C code implementation

>50% of peak performance (SSE)

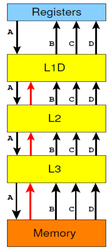
ALU saturation pattern

Starting point: Roofline model [5]

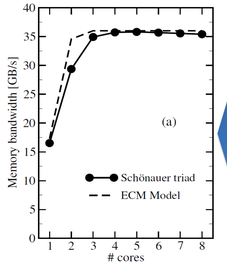


Refinement: Take data transfer into account

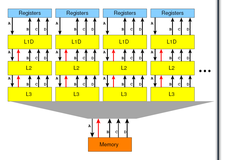
ECM Model [4]



Assumption f. bottleneck saturation

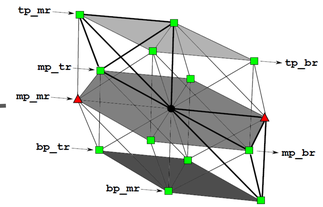


Prediction of saturation behavior



Example: 3D 15-point red-black stencil in MG solver [6]

```
for (int i=1; i < (tsize-j-k-1); i=i+2) {
  u[mp_mr+i] = c[0] * (
    -c[1] *u[mp_mr+i+1] - c[2] *u[mp_tr+i-1] -
     c[3] *u[mp_tr+i]   - c[4] *u[tp_br+i]   -
     c[5] *u[tp_br+i+1] - c[6] *u[tp_mr+i-1] -
     c[7] *u[tp_mr+i]   - c[8] *u[bp_mr+i]   -
     c[9] *u[bp_mr+i+1] - c[10]*u[bp_tr+i-1] -
     c[11]*u[bp_tr+i]   - c[12]*u[mp_br+i]   -
     c[13]*u[mp_br+i+1] - c[14]*u[mp_mr+i-1] +
     f[mp_mr+i] );
}
```



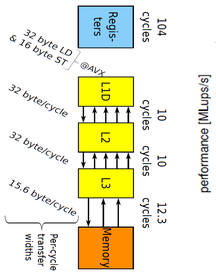
Roofline analysis

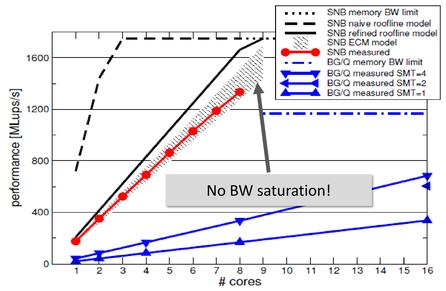
Memory-bound algorithm!

Memory BW saturation pattern

ECM Model analysis using IACA

ALU saturation pattern





References

[1] J. Treibig, G. Hager, and G. Wellein, "Performance patterns and hardware metrics on modern multicore processors: Best practices for performance engineering." in Euro-Par Workshops, ser. Lecture Notes in Computer Science, I. Caragiannis, M. Alexander, R. M. Badia, M. Cannataro, A. Costan, M. Danelutto, F. Desprez, B. Krammer, J. Sahuquillo, S. L. Scott, and J. Weidendorfer, Eds., vol. 7640. Springer, 2012, pp. 451– 460.

[2] J. Treibig and G. Hager, "Introducing a performance model for bandwidth-limited loop kernels," in Parallel Processing and Applied Mathematics, ser. Lecture Notes in Computer Science, R. Wyrzykowski,J. Dongarra, K. Karczewski, and J. Wasniewski, Eds. / Heidelberg, 2010, vol. 6067, pp. 615–624.

[3] J. Treibig, G. Hager, H. G. Hofmann, J. Hornegger, and G. Wellein: "Pushing the limits for medical image reconstruction on recent standard multicore processors". International Journal of High Performance Computing Applications 27(2), 162–177 (2013).

[4] G. Hager, J. Treibig, J. Habich, and G. Wellein: Exploring performance and power properties of modern multicore chips via simple machine models. Submitted. Preprint: arXiv:1208.2908

[5] S. W. Williams, A. Waterman, and D. A. Patterson, "Roofline: An insight- ful visual performance model for floating-point programs and multicore architectures," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2008-134, Oct 2008. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-134.html

[6] G. Gmeiner, U. Ruede, H. Stengel, C. Waluga, B. Wohlmuth "Performance and Scalability of Hierarchical Hybrid Multigrid Solvers for Stokes Systems", in preparation

[7] Jan Treibig, Georg Hager, and Gerhard Wellein. Likwid: A lightweight performanceoriented tool suite for x86 multicore environments. In Wang-Chien Lee and Xin Yuan, editors, ICPP Workshops, pages 207–216. IEEE Computer Society, 2010