ERLANGEN REGIONAL COMPUTING CENTER

Efficient numerical simulation on multicore processors (MuCoSim) SS 2019

Prof. Gerhard Wellein

Department für Informatik & HPC Services
Regionales Rechenzentrum Erlangen (RRZE)

https://moodle.rrze.uni-erlangen.de/course/view.php?id=395
(see also univis)

Mission

We
care
about
performance!

Mission

Optimization & Parallelization on all modern compute architectures

→Benchmarking & Performance Measurement

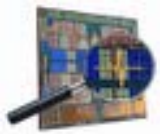
→Understand interaction between code & hardware

→Performance modelling: Roofline model & ECM model

This is ours!

→Performance tools:

likwid – lightweight performance tools
(https://github.com/RRZE-HPC/likwid)



kerncraft - Loop Kernel Analysis & Performance Modeling Toolkit
(https://github.com/RRZE-HPC/kerncraft)

Mission

Performance optimization, performance modeling, parallelization for

- Multi-core CPUs: core, socket, node & large scale 100,000+ cores
- GPGPUs: single devices and cluster
- Many-core CPUs: Intel Xeon Phi

- Our group:
  - 6 senior scientists (incl. RRZE) (GW/GH/TZ/MM/JE/KN)
  - 6 PhD students (TG/JHa/JHo/CLA/DE/AA)
  - 2 Master/Bachelor students (HA/JL)

- We operate the compute resources at FAU

Machines

- RRZE Testcluster („Playground“)
  - 18-core Broadwell / 20-core Skylake / 24-core AMD Epyc / 32-core ARM TX2
  - nVIDIA V100 & NEC SX-Aurora Tsubasa

- RRZE production machines (Infiniband/Ominpath Interconnect):
  - Emmy:
    544 nodes Intel Ivy Bridge (2x 10 cores/node) → 10.880 cores
    - + 8 nodes with 2 x NVIDIA K20 + ~~8 nodes with 2 x Intel Xeon Phi / KNC~~
  - Meggie:
    728 nodes Intel Broadwell (2x10 cores / node) → 14.560 cores

- Access to external machines
  - LRZ Garching: Intel Skylake Cluster (26.9 Pflop/s; 311,040 cores;)
    Most powerful CPU-only cluster in the world!

ERLANGEN REGIONAL COMPUTING CENTER

MuCoSim SS 2019

Seminar topics

What to do in the seminar

- Get familiar with some code (C/C++/Fortran)
- Carefully measure and report (performance) numbers for (various) modern compute device(s)
- Implement (small) code modifications and measure their impact
- Do (simple) performance model if necessary/possible
- Extend feature set of tools (some topics)

What we expect

- Basic knowledge of C, C++ or FORTRAN
- Basic knowledge in Linux shell usage
- Basic knowledge in OpenMP parallelization (some projects)
- Basic knowledge in Python
- You can use some texteditor (vi, vim, emacs,...)

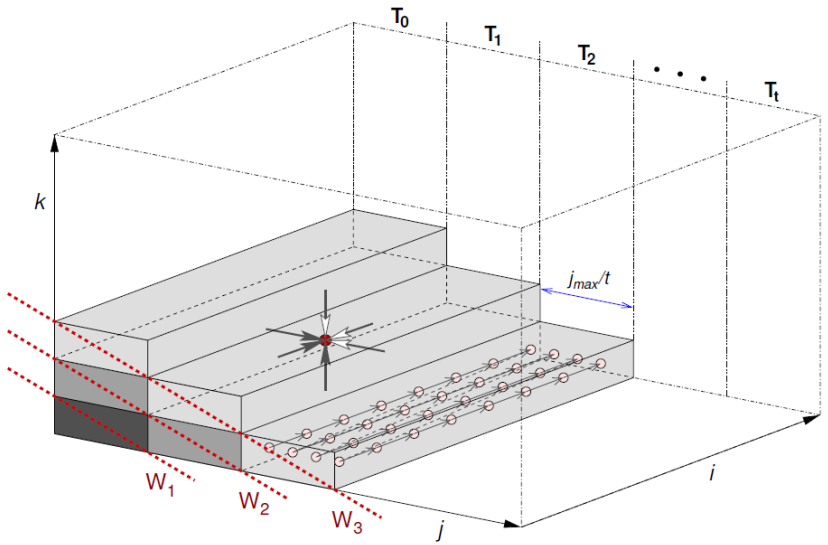
- You need to actively participate in two hands-on sessions where you learn
  - how to access and use our machines,
  - how to compile and run a code,
  - how to use our benchmarking and analysis tool likwid

- Hands-on Sessions: 30.4. (15:00-18:00) & 7.5. (16:30-...)

OpenMP tasking with dependencies (Georg Hager)

Parallelizing and analyzing a Gauss-Seidel algorithm with OpenMP and tasking

- GS can be parallelized easily with pipelined parallel execution ("PPP" → PTfS lecture)
- The abundance of barriers may cause performance degradation
- Idea: Parallelize GS with OpenMP tasks, specifying dependencies between blocks
- Tasks
  - Implementations: PPP and tasking in 2D and 3D
  - Performance analysis and Roofline modeling, analysis of tasking/barrier overhead
  - Try Intel and GCC OpenMP runtimes
  - Can the tasking approach outperform the PPP solution at all? How?



Stepanov Reloaded (Georg Hager)

Designing an improved C++ abstraction benchmark

- Standard benchmark: vector sum

```
for(i=0; i<2000; i++)
  s += a[i];
```

- C++ Stepanov test: 13 versions of this with increasing level of abstraction
  - "When does the compiler stop seeing through the abstractions?"
  - Very well handled by modern compilers
  - We need something better and more modern

- Proposal: Simple 2D 5-point stencil iteration
  - … but I'm open for suggestions

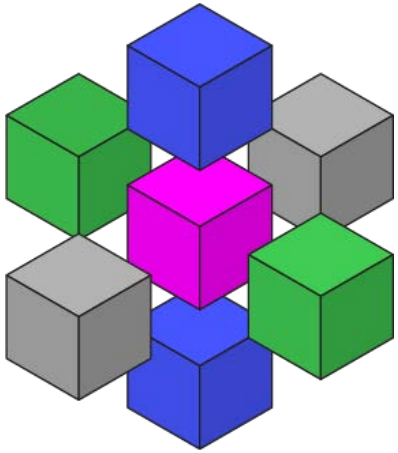
Stencils on Tsubasa (Georg Hager)

NEC Aurora “Tsubasa” Vector Engine

- Native vector processor architecture
- Memory bandwidth ≈ 1 TB/sec
- 8 cores
- C/C++/Fortran vectorizing compilers, OpenMP support

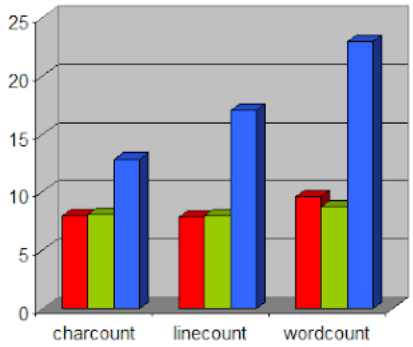
Task: Port and investigate simple stencil algorithms

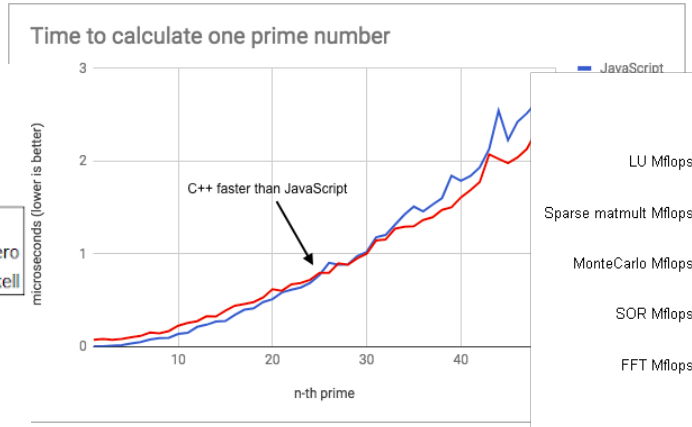
- 2D 5-pt
- 3D 7-pt
- Roofline model
- Benchmarking vs. problem size & # of cores
- Loop blocking, unrolling

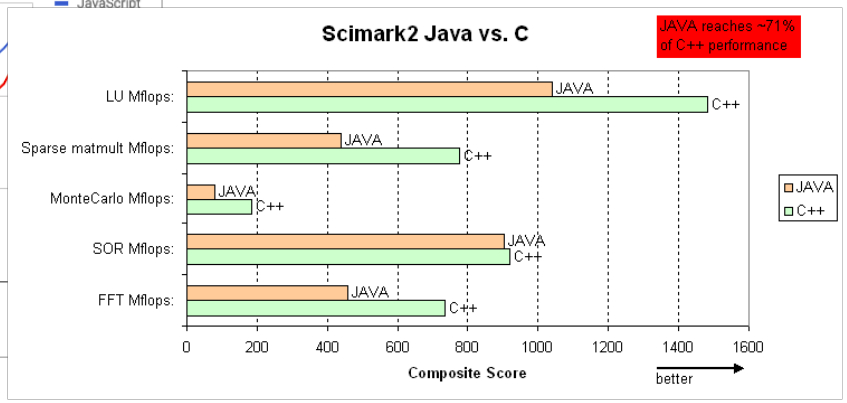


Performance of “modern” languages (Dominik Ernst)

- HPC codes mostly implemented in C, C++ and Fortran
- There is a plethora of much hipper languages
- Claims of “C-like performance”, supported by opaque benchmarks
- We do our own benchmarks!
- Task: Write a numerical benchmark of relevance in HPC (e. g. Jacobi stencil update)
- Benchmark a selection of ‘modern’ languages, e. g. Go, Rust ,Julia, Python, Swift, ~~Perl~~, ~~Java~~… (add new hip language here)
- What is the ‘idiomatic’ performance? How far can you bend a language for performance?







Check the threading models provided by programming languages (Thomas Gruber)

- Many programming languages provide threading functionality
- Write a simple multithreaded code in different languages
- How much can threading be controlled?
  - Affinity?
  - Local data?
- Does the language use software threads or OS threads?
- Can it be used for distributed (multi-node)?
- Possible languages: C++11 threads, Julia, Rust, Swift, Go, …

Analysis of MiniMD (Jan Eitzinger)

`MiniMD` – Molecular Dynamics Proxy App

- App. 3000 lines, C++
- Extracted from LAMMPS Molecular dynamics application code

Tasks:

- Perform application benchmarking with different compilers and SIMD variants
- Create a runtime profile
- Perform a Hardware Performance Monitoring profile
- Analyse and quantify the efficiency of SIMD usage

The Bandwidth Benchmark (Jan Eitzinger)

https://github.com/RRZE-HPC/TheBandwidthBenchmark

Minimal main memory bandwidth microbenchmarks with multiple streaming kernels.

Tasks:

- Run benchmark on various generations of Intel and AMD processors
- Compare the reported values with profiling results using **`likwid-perfctr`**
- Document and discuss the results

Analyze dense matrix-vector multiplication (Thomas Gruber)

- Dense MVM is a common operation in HPC
- Often part of HPC courses

Task:

- Establish simple performance model(s) for dMVM
- Perform hardware measurements using LIKWID on different CPUs
- Compare results with model and make refinements
- Propose optimizations for naïve algorithm

Analyze branch prediction systems of modern architectures (Thomas Gruber)

- Common codes contain a lot of conditions → branches
- CPUs try to predict outcome to speculatively execute code sections
- CPUs provide measurement facilities for branching

Task:

- Analyze how detailed branching can be analyzed
- How does mispredictions limit code execution (stalls, pipeline drains, …)

Top-Down Microarchitecture Analysis Method

by Ahmad Yasin (Intel) (Thomas Gruber)

- Analysis of codes on cycle basis
- Integrated into VTune Amplifier and perf
- Use common codes from benchmark suites

TASK:

- Compare TMAM for common codes on a set of architectures
- What are the reasons for result changes if using another arch.?
- If a code has a limitation, how well it can be found with TMAM?

Adding and testing PAPI to likwid-bench (Thomas Gruber)

- PAPI provides an abstraction layer for various measurement facilities (e.g. hardware performance counter)
- likwid-bench is a micro-benchmarking suite with assembly kernels

Task:

- Add PAPI calls to likwid-bench for common measurement groups (L2, L3, FLOPS_DP, FLOPS_SP, …)
- Compare measurements of PAPI with LIKWID measurements

Kerncraft: Stand-alone Benchmark and PhenoECM (Julian Hammer)

Task:

- Build a script which does the same as

  `kerncraft -p Benchmark`
- Make it more versatile to work with any loop code
- Remove necessaty for Kerncraft‘s code analysis

Software and techniques involved:

- Python, git, likwid, assembly

OSACA & asmbench: Complete Database (Julian Hammer)

- We are currently missing some AVX512 and all AMD data.

Task:

- Use asmbench to complete OSACA instructions databse.

Software and techniques involved:

- Python, git, llvm, uops.info, assembly

Extract Data Accesses Pattern from Assembly (Julian Hammer)

- Kerncraft needs data access patterns (array offsets)
- A technique to extract such patterns from assembly, would be useful in making KC more versatile

Software and techniques involved:

- Python, git, assembly

Interface pycachesim with Valgrind (or PIN) (Julian Hammer)

- Valgrind and PIN allow live capturing of memory addresses accessed
- Using these accesses, cache hits and misses can be derived with pycachesim and related to code lines

Software and techniques involved:

- Python, git, C, Valgrind/PIN