Autotuning Benchmarking Techniques: A Roofline Model Case Study

Jacob O. Tørring, Dr. Jan Christian Meyer, Prof. Anne C. Elster

Department of Computer Science
Norwegian University of Science and Technology (NTNU)

Motivation

— Scheduling and hardware selection
— Modelling the performance of architectures [1]
— Roofline model [2] to compare systems
— Theoretical peak performance is often available from vendors
— However practical peak performance is often far lower
— Find peak practical performance through autotuning benchmarks

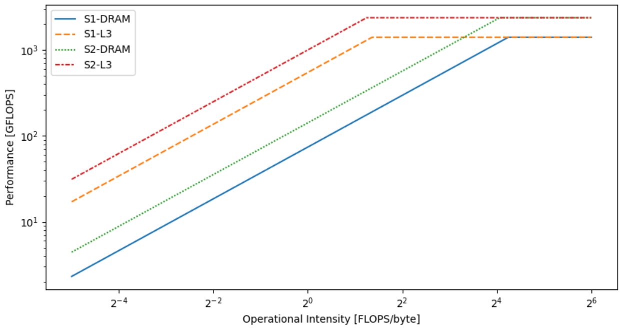
Contributions

— Tool for automatically generating practical performance Roofline models using high performing autotuned benchmarks
— Significant search time improvements from autotuning benchmarking techniques, up to 116.33x
— General autotuning benchmarking techniques that can be applied to any autotuning application

Outline

Roofline model

- Visual performance model
- Developed by Williams et al. [2]
- Operational Intensity
  OI = $\frac{operations}{byte}$
- $F_\alpha(OI) = \min(B_\alpha \cdot OI, F_p)$
- High OI = Peak Compute Performance (F_p)
- Low OI = Peak Memory Performance (B_α)
- DRAM vs L3 Cache



Roofline model: Benchmarks and Related Work

- F_p is usually given by the High OI benchmark DGEMM
  - Double-precision GEneral Matrix Multiply (DGEMM)
  - $C \leftarrow \alpha AB + \beta C$
  - $A = n \times k$, $B = k \times m$, $C = n \times m$, $\alpha = 1.0$, $\beta = 0.0$
- B_α is usually given by the Low OI benchmark TRIAD from STREAM [3]
  - Double-precision vector addition
  - TRIAD: $C \leftarrow A + \gamma B$, $\gamma = 1.0$
- Intel Advisor Tool: Proprietary and limited to Intel processors
- Ilic and Denoyelle [4], as well as Marques et al. [5]

Autotuning Search space

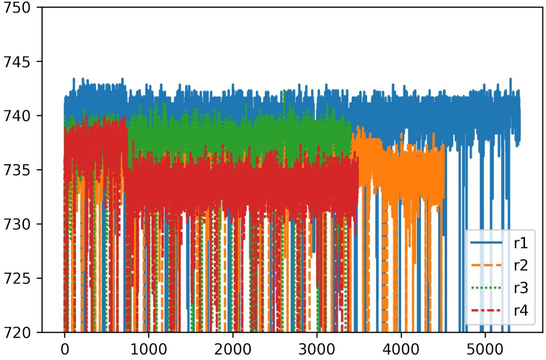
- Find F_p through autotuning DGEMM computations.
- Find the optimal matrix dimensions n, m, k to maximize hardware performance
- Start by constraining the search space
  - With steps of power of 2 from 64 to 4096 for n and m and 2 to 2048 for k
  - DGEMM: $S = n \times m \times k, \quad |S| = 7 \cdot 7 \cdot 11 = 539$
  - Through experimentation this was reduced further.
  - From 512 to 4096 for n and m and 64 to 2048 for k.
  - The cardinality is thus $4 \cdot 4 \cdot 6 = 96$.

Autotuning Techniques

- Sample cost is low $\implies$ balancing overhead of advanced techniques vs. gathering more samples
- Search space is small $\implies$ random search might not be ideal compared to exhaustive search
- Exhaustive search is an easy and high performing alternative in this scenario
- It also clearly illustrates the benefits of autotuning benchmarking techniques

Autotuning Benchmarking

- Iteration: The program executes the DGEMM/TRIAD operations several times
- Invocation: The benchmarking program is executed several times
- Take the mean of all iterations and all invocations



Autotuning Benchmarking

— Stop conditions
1. Total time threshold for each invocation of the benchmarking process
2. Maximum number of iterations of the benchmark for each sample

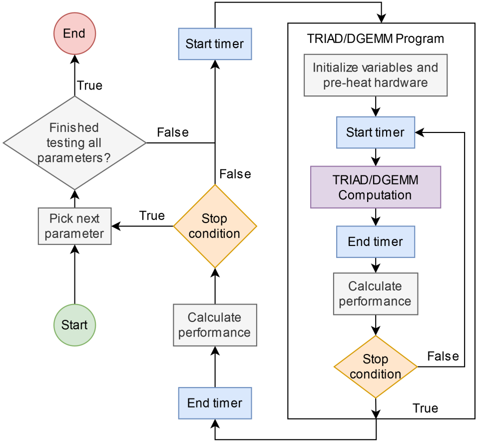
— *Early* stopping conditions
- Construct a confidence interval of the mean value for each benchmarked sample
- Continually update the confidence interval throughout the benchmarking process
- Only used as a heuristic, due to the normality assumption

— This enables early stopping of the benchmarking when
3. The mean has achieved a sufficient accuracy
   $\frac{upper}{mean} - 1 < \Delta$, e.g. $\Delta = 0.01$, upper confidence interval is 501, mean value is 500, then $\frac{501}{500} - 1 < 0.01$
4. The confidence interval's upper bound is lower than the previously best sample
   $upper < best$

Autotuning Benchmarking

- Welford's Online variance algorithm
  - Constant time variance calculation regardless of iteration count
  - Only need to store two variables (mean and variance)
- Future work includes other data structures and other statistical methods as heuristics

Autotuning Pipeline

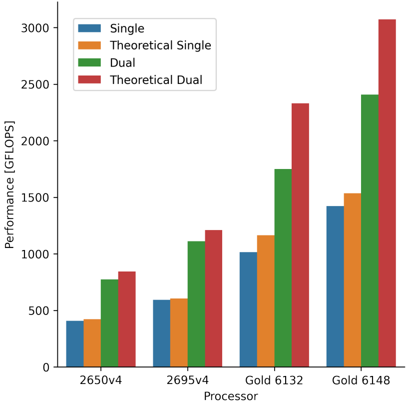


Experimental Setup

- Experiments are conducted on the Idun [6] cluster at NTNU
- Tested on dual-socket Intel systems
  - With 2650v4, 2695v4, Gold 6132 and Gold 6148 CPUs
- Theoretical peak compute performance: $F_t = freq \cdot cores \cdot AVX_{type} \cdot AVX_{units} \cdot CPUs$
- Theoretical peak memory performance: $B_t = freq \cdot channels \cdot \frac{bytes}{cycle}$
- Maximum 200 Iterations, 10 invocations, 10s timeout for each invocation and a 99% CI delta of 1%.
- Executed using Intel's MKL BLAS implementation and SLURM

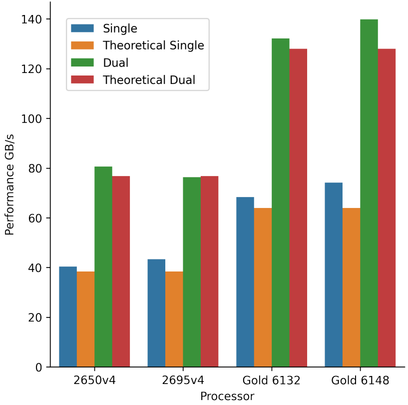
Results: DGEMM Performance

- Intel's related work [7] was able to achieve 52.08% of theoretical maximum
- Autotuned dual-socket results range from 75.13%–91.93%
- Autotuned single-socket results range from 87.20%–98.06%
- AVX512 workloads are usually clocked lower



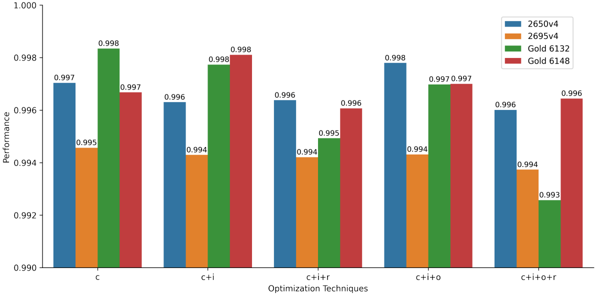
Results: TRIAD Performance

- Autotuned DRAM dual-socket results range from 99.37%–109.25%
- Autotuned DRAM single-socket results range from 105.26%–115.90%
- We believe that the performance exceeding 100% is due to the effect of cache on memory performance



Results: Early stopping optimizations

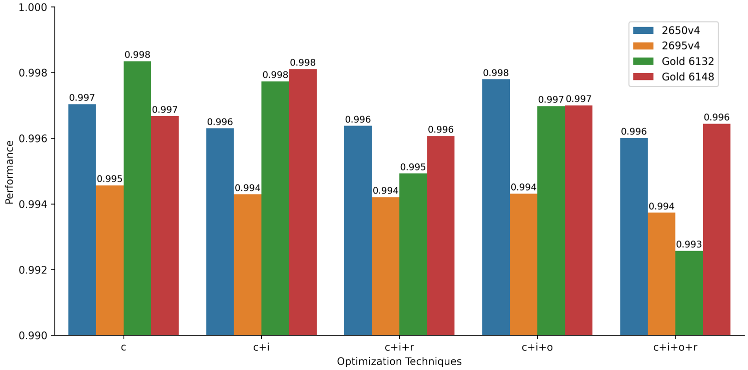
- "c": Stop condition 3 (absolute CI)
- "c+i": Additionally stop condition 4 (relative CI) applied to "inner" iteration loop
- "c+i+r": Reversal of search order
- "c+i+o": Stop condition 4 (relative CI) applied to iteration and "outer" invocation loop
- "c+i+o+r": Reversal of search order



For Intel 2695v4 we applied a lower bound on stop condition 4 of 100 iterations, to ensure that it could find the highest performing configurations, that peaked late into the iteration count. Full details and exploration of this is available in the paper

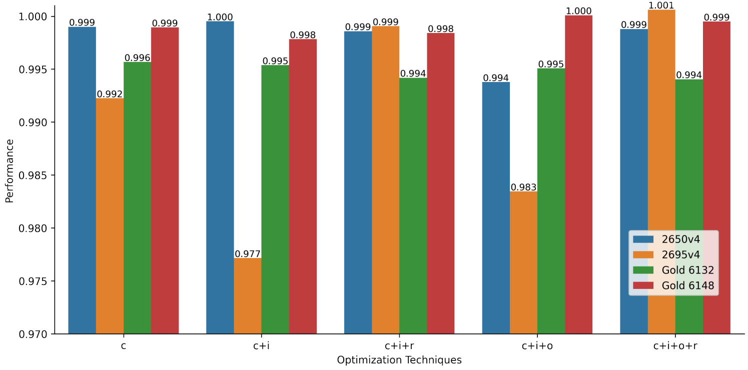
Results: Single socket performance accuracy

- Single socket performance accuracy
- 99.3% to 99.8% compared to non-optimized benchmarking results

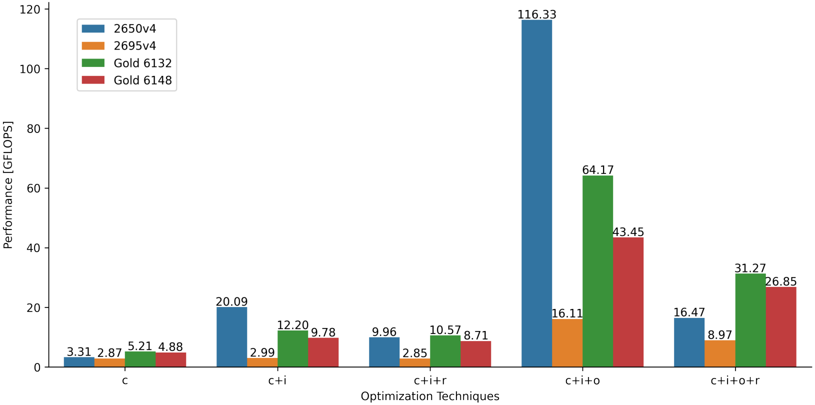


Results: Dual socket performance accuracy

- Dual socket performance accuracy
- 98.3% to 100.1% compared to non-optimized benchmarking results
- The highest performing sample for 2695v4 scales late into the iteration count

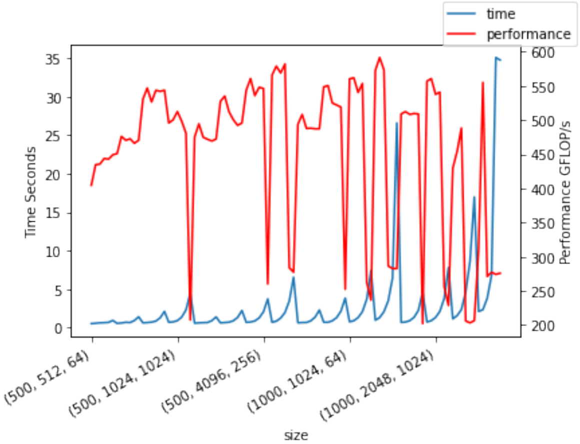


Results: Optimizations Performance



Discussion: Optimizations

- The performance of stop condition 4 is dependent on the search order of the autotuning process
- Samples with low performance and a high cost early in the search cannot be skipped due to lack of previous high performance alternatives
- Search should therefore try to target low cost samples initially



Conclusion

- Tool for automatically generating practical performance Roofline models using high performing autotuned benchmarks
- Significant search time improvements from autotuning benchmarking techniques, up to 116.33x
- General autotuning benchmarking techniques that can be applied to any autotuning application

Future work

- Benchmarking L2 and L1 cache using more accurate benchmarks and measurements
- Changing the data structure and how we compare relative performance between samples, to include more information than the mean value of the sample
- This change can potentially lead to more accurate predictions for when it is safe to terminate

Thank you for listening!

Contact information
Jacob O. Tørring: jacob.torring@ntnu.no
Jan Christian Meyer: jan.christian.meyer@ntnu.no
Anne C. Elster: elster@ntnu.no

References I

[1] Jan Christian Meyer. *Performance Modeling of Heterogeneous Systems*. eng. Accepted: 2014-12-19T13:39:21Z. Norges teknisk-naturvitenskapelige universitet, Fakultet for informasjonsteknologi, matematikk og elektroteknikk, Institutt for datateknikk og informasjonsvitenskap, 2012. ISBN: 978-82-471-4015-4. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/253074 (visited on 02/10/2021).

[2] Samuel Williams, Andrew Waterman, and David Patterson. *Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures*. en. Tech. rep. 1407078. Sept. 2009, p. 1407078. DOI: 10.2172/1407078. URL: http://www.osti.gov/servlets/purl/1407078/ (visited on 08/10/2020).

[3] John D. McCalpin. *STREAM: Sustainable Memory Bandwidth in High Performance Computers*. Tech. rep. A continually updated technical report. http://www.cs.virginia.edu/stream/. Charlottesville, Virginia: University of Virginia, 1991-2007. URL: http://www.cs.virginia.edu/stream/.

References II

[4] Aleksandar Ilic, Frederico Pratas, and Leonel Sousa. “Cache-aware Roofline model: Upgrading the loft”. en. In: *IEEE Computer Architecture Letters* 13.1 (Jan. 2014), pp. 21–24. ISSN: 1556-6056. DOI: 10.1109/L-CA.2013.6. URL: http://ieeexplore.ieee.org/document/6506838/ (visited on 08/12/2020).

[5] Diogo Marques et al. “Application-driven Cache-Aware Roofline Model”. en. In: *Future Generation Computer Systems* 107 (June 2020), pp. 257–273. ISSN: 0167-739X. DOI: 10.1016/j.future.2020.01.044. URL: http://www.sciencedirect.com/science/article/pii/S0167739X19309586 (visited on 08/12/2020).

[6] Magnus Själander et al. “EPIC: An Energy-Efficient, High-Performance GPGPU Computing Research Infrastructure”. In: *arXiv:1912.05848 [cs]* (Dec. 2020). arXiv: 1912.05848. URL: http://arxiv.org/abs/1912.05848 (visited on 01/18/2021).

[7] Ying Hu and Shane A Story. *Tips to Measure the Performance of Matrix Multiplication Using Intel®...* en. Dec. 2017. URL: https://www.intel.com/content/www/us/en/develop/articles/a-simple-example-to-measure-the-performance-of-an-intel-mkl-function.html (visited on 08/10/2020).