Parallel Scaling Properties from a Basic Block View
Melanie Kambadur, Kui Tang, Joshua Lopez, Martha A. Kim

The goal:
Narrow down the source of problematic parallel program scaling.

Micro-scaling metrics expose a broad range of performance flaws by reporting the quantitative and qualitative scaling properties of a program’s basic blocks. The metrics capture the effects of hardware, operating systems, thread models, and inputs, and reveal performance issues about arbitrary regions of interest, including individual basic blocks, functions, algorithms, or critical sections. These software-centric metrics can be collected at runtime with minimal programmer effort, require no hardware modifications, and are not language or thread library specific.

Isolating scaling deficiencies in benchmarks

Micro-scaling metrics at the application level show coverage of scaling deficiencies. These code coverage breakdowns hint at the prevalence of different kinds of performance deficiencies throughout a code base. For example, if there are a small number of poorly scaling blocks in a program, then for a few lines of code most of the scalability issues.

Micro-scaling metrics show that badly scaling program segments do not align well with functions. Many functions in a SPEC cgsf benchmark and other applications contain basic blocks with a variety of scaling properties. For example the printf() function contains blocks that exhibit five different qualitative types of scaling behavior:

1. They also show that blocks are responsible for a large percentage of runtime even as parallelism scales up. The table at left lists the relative time consumed by the top ten hottest blocks out of the whole program time. While the ten hottest blocks are not always the same at 2 and 16 threads, it is true that at both degrees of parallelism only a few blocks take up most of the program runtime.

Micro-Scaling Metrics in source code. It can be tricky to reason about basic block boundaries in software. To make micro-scaling metrics understandable at a high level, debug annotated assembly code can be used to propagate the qualitative metric categorizations into source code annotations. For example, we might color perfectly scaling blocks green, moderately scaling blocks blue, poorly scaling blocks yellow, non-scaling red, etc.

Methodology Details

Machine Specifications. The hardware platform for all experiments was a Dell Precision T5500 workstation with two quad core processors (Intel X5550, 2.66 GHz, 8M last level cache) and 24 GB of RAM.

Benchmarks. We used a selection of applications in the Parsec parallel benchmark suite [2]. These applications are non-scientific, written in C and C++, parallelized using pthreads, and have a variety of scaling behaviors. We ran each application twenty times with 1, 2, 4, 8, and 16 threads.

DIA-Time Approximation. Block execution counts are not always sufficient to determine runtime—differing block lengths and instructions cause block execution time to vary significantly. So, we use a model called Dynamic Instruction Approximate, or DIA-time. Blocks’ dynamic instruction counts are linearly weighted by measured average Intel SSE instruction latencies, and by the degree of parallelism exhibited by the program as a whole during a block’s execution.

The Future of Micro-Scaling Metrics

• Integration into existing performance analysis tools. Existing parallel performance analysis either look for specific kinds of inefficiencies in software or find a broader range of inefficiencies by monitoring ineffective use of hardware and thread resources over time. In the worst case, the thread or time-based resource usage patterns have to be mapped back to the source code. Instead, micro-scaling metrics monitor parallel performance problems in the program’s point of view, so that any scaling deficiencies found are already tied to the source code. We think this is new, flexible, fine-grained analysis complements existing techniques, and it would be beneficial to add micro-scaling metrics to existing performance analysis tools for software engineers.

• Quantifying the ability of micro-scaling to narrow down scaling deficiencies. In future work, we plan to concisely measure the degree to which micro-scaling metrics narrow down scalability problems in source code. We plan to examine this both in benchmarks and real-world applications.

Acknowledgements

This work has been made possible through the generous support of the National Science Foundation (NSF-111713).

References


ARCADE Lab, Columbia University, New York, NY
http://arcade.cs.columbia.edu