Parallel Program Scaling Analysis using Hardware Counters

Shobhit Jagga*
Indian Institute of Technology Kanpur
Kanpur, India
shobhitj@iitk.ac.in

Preeti Malakar
Indian Institute of Technology Kanpur
Kanpur, India
pmalakar@iitk.ac.in

ABSTRACT

We present a lightweight library that automatically collects several hardware counters for MPI applications. We analyze the effect of strong and weak scaling on the counters. We first correlate the counter values obtained from each process count, and then cluster the counters to identify counters that are affected similarly due to scaling. We noted that the effect of last-level cache misses is more pronounced for some applications such as miniFE.

KEYWORDS

Performance analysis; hardware counter; code characterization

ACM Reference Format:
Shobhit Jagga and Preeti Malakar. 2021. Parallel Program Scaling Analysis using Hardware Counters. In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '21), June 21–25, 2021, Virtual Event, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3431379.3464453

1 INTRODUCTION

Large-scale parallel applications are ubiquitous in today's petascale era. Such applications are routinely executed on different core counts, often with different problem sizes during each run, as may be required to realize the science goals. Therefore, it is imperative to study, and analyze the scaling behavior of applications so that the production runs are more efficient. Scaling runs show the efficiency of applications on a given system.One of the approaches for performance analyses is profiling applications using hardware counters [1, 4, 6]. Counters such as cache misses, load/store misses, and stalled CPU cycles due to memory access provide an understanding of the behavior of the application on a given system.

Our contribution: We propose using hardware counters for code characterization. We correlate hardware measurements with application scaling to analyze the effect of scaling on the counters. Since the number of hardware counters are typically large, we correlate different counters and then perform clustering to identify counters that affect scalability similarly. This may be used to identify bottlenecks in parallel programs due to memory, computation and communication. In this work, we focus on the first two. We used the Performance Application programming interface (PAPI) [5] that provides an interface for hardware counter collection. However, all desirable/available counters cannot be collected simultaneously due to counter conflicts [4]. Multiplexing provides a resolution to this via event sets [4], however this may be inaccurate and insufficient due to limited registers [6], especially if we multiplex between several counters. We have developed a library that automatically collects the hardware counters via multiple runs of the application using a configuration file to collect non-conflicting counters at the same time. We used the MPI Profiling Interface (PMPI) for automatic hardware counter collection using compile-time instrumentation of MPI codes. We used 32 hardware counters out of the 59 available counters.

*Student Author

2 RELATED WORK

Performance modeling techniques are widely used to avoid overheads and potential performance deviations arising due to application instrumentation. Analytic performance modeling [2] attempts to characterize application behaviour by analyzing the algorithms and source code. However, this requires understanding the code. There has been work on predictive performance models to predict performance and scalability bottlenecks. Ding et al. [1] propose a resource based modeling approach by hardware counter collection to predict the scaling issues and run-time of the application. Lively et al. [3] model the power consumption alongside performance modeling to draw better insights into power-performance tradeoffs for multi-core systems. In contrast, we propose correlating and clustering counters to identify those that affect scalability similarly.

3 APPLICATION SCALING ANALYSIS

We use hardware counters to analyze the scaling performance of applications. It is infeasible to analyze all counters that may be present in some systems ($\sim$ 400). We have built a library to automatically collect hardware counters during an application run, and analyze the impact of strong and weak scaling on the counter values. Additionally, there may be related counters that show similar behaviour with scaling. Therefore, we first correlate the counters (normalized) and then perform clustering to determine their effect on application behaviour. This also helps to introspect into counters in a single cluster to identify potential bottlenecks arising due to memory and computation. This may help identify the cause of performance slowdown by analyzing the behaviour of counters such as increased cache misses leading to scalability bottlenecks. The required hardware counters are specified via a configuration file, such that all of them can be automatically collected via multiple runs of the application; a set of non-conflicting counters being collected during every run. We have built a library that uses the MPI Profiling Interface (PMPI) to instrument the application. We initialize the PAPI library and insert the PAPI code in the MPI_Init() function of our library. We accumulate the counters in the MPI_Finalize() function of our library. Thus there is no source code modification required to collect the counters.

4 EXPERIMENTS AND RESULTS

We used four benchmarks – HPCG, miniFE, miniGhost and XSBench to perform strong and weak scaling runs on up to 40 2.5 GHz Intel Xeon Gold 5215 cores (3 nodes). The applications were

instrumented with our library to collect 32 hardware counters, such as L1, L2 and L3 cache statistics (misses, hits), memory stalls, TLB statistics, instruction types (load, store), etc. We used average of two runs. The counters for all cores were first averaged to represent a single counter value across all processes. This is done for 10, 20, 30 and 40 processes for both strong and weak scaling runs. A $32X32$ correlation matrix is then generated by computing correlations for all counter pairs from the 4 values per counter corresponding to the 4 process counts (10 – 40). This is done to observe similar behavior of counters when scaling an application. This matrix is then used to hierarchically cluster the counters to find counter clusters that affect the application scaling similarly.

Figures 1 and 2 show the strong and weak scaling correlation matrices for the four applications. The last counter (last row, last column) corresponds to total number of cycles for the application run. We observe that the correlation matrix of miniFE is

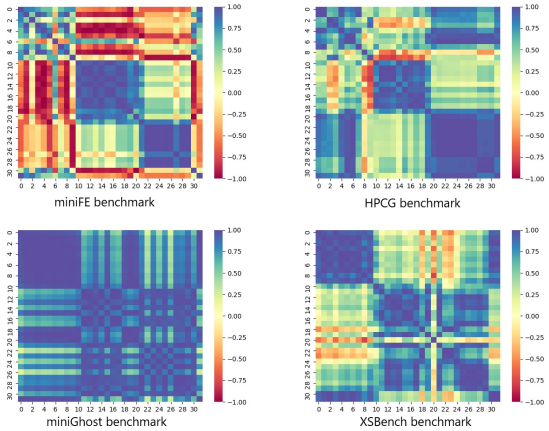

Figure 1: Correlation matrix of 32 hardware counters for four mini-applications from strong scaling runs on 10 – 40 cores.

distinct. There is strong positive correlation among some of the counters 10 – 18. These are related to L1 cache load/store misses, TLB misses, cache line intervention and shared cache line request counters. We observe that L3 total cache misses (counter #8) has strong negative correlations with some of the counters 10 – 18. Specifically we noted that L3 total cache misses (PAPI_L3_TCM) decrease till 30 processes and then increase at process count 40, while the number of TLB misses keep decreasing. We also noted that the number of cache snoops (PAPI_CA_SNP) first decrease and then increase at process count 40, thus the correlation between PAPI_L3_TCM and PAPI_CA_SNP is 0.5. This may be due to the unstructured grid miniFE application. We note that decreased data size due to strong scaling helped with better data locality initially, however, increasing using more cores further increased the load on shared cache (evident from increased snoops, shared cache line requests) and resulted in more private cache misses (evident from more L1 and L2 total cache misses at process count of 40). This may be due to NUMA-unaware placement of MPI ranks, we will investigate this further in future. We observe similar interesting trends in the other applications. We also noted that the number of total cycles is more strongly correlated with the private cache misses in case of strong scaling. This indicates that the effect of data-size reduction on caches outweighs other counters' variations and is instrumental in determining the application run-time and performance. We observe clearly that the first 10 and the last 8

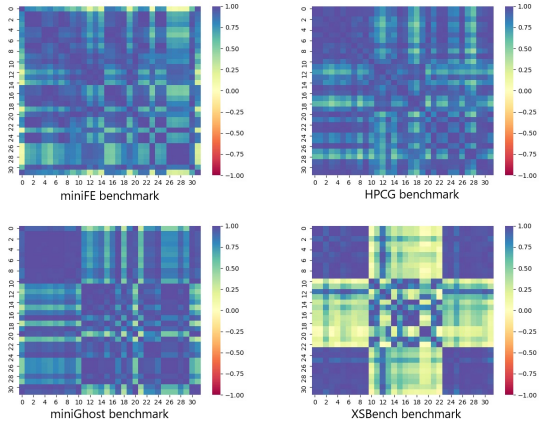

Figure 2: Correlation matrix of 32 hardware counters for four mini-applications from weak scaling runs on 10 – 40 cores.

counters are strongly correlated in weak scaling XSBench runs, whereas there is no perceivable correlation among the rest. Most applications show similar trends of the counters. On inspecting the counters, and the performance of weak scaling runs, we noted that almost all counters showed increasing trends, including the total number of cycles. Thus the applications scaled poorly when weak scaled. We also observe from the Figures 1 and 2 that many counters have positive correlation in one and negative in the other. For example, L2 data cache misses, PAPI_L2_DCM (counter #6) and L2 total cache accesses, PAPI_L2_TCA (counter #30) are strongly negatively correlated in case of miniFE strong scaling whereas they are strongly positively correlated in case of miniFE weak scaling. This indicates that data size per node impacts the counter trends in miniFE.

5 CONCLUSIONS AND FUTURE WORK

Our library provides an automatic way of collecting several hardware counters without source-code modifications. We presented the effect of scaling on 32 counters, and noted different counter trends in four mini-applications for strong scaling on 10 – 40 cores. Potential future work is to use this approach for more real world applications with different data sizes and analyze the effect of NUMA-aware placements with the help of these counters.

REFERENCES

[1] N. Ding and et al. 2019. Using hardware counter-based performance model to diagnose scaling issues of HPC applications. *Neural Comput Applic* (2019).
[2] Darren J Kerbyson and Philip W Jones. 2005. A performance model of the parallel ocean program. *The International Journal of High Performance Computing Applications* (2005).
[3] Charles Lively and et al. 2012. Power-aware predictive models of hybrid (MPI/OpenMP) scientific applications on multicore systems. *Computer Science-Research and Development* 27, 4 (2012), 245–253.
[4] J. M. May. 2001. MPX: Software for multiplexing hardware performance counters in multithreaded programs. In *Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001.*
[5] Dan Terpstra and et al. 2010. Collecting Performance Data with PAPI-C. In *Tools for High Performance Computing 2009.*
[6] Y. C. Wang, J. Wang, J. K. Chen, S. C. Zuo, X. M. Su, and J. Lin. 2020. NeoMPX: Characterizing and Improving Estimation of Multiplexing Hardware Counters for PAPI. In *2020 IEEE International Conference on Cluster Computing (CLUSTER).*