# Performance Analysis of HPCG Benchmark

Minwoo Ahn[1], Nayeon Kim[2], Chunkyu Park[2], and Jinkyu Jeong[2]

[1] *Sungkyunkwan University, Suwon 2066, Republic of Korea*

[2] *Yonsei University, Seoul 03722, Republic of Korea*

Corresponding author (Electronic mail: jinkyu@yonsei.ac.kr)

In this paper, we present the performance analysis results of the large-scale computational application, the HPCG benchmark, along with optimization strategies. Profiling is conducted using the recently introduced sampling-based bottleneck analysis tool, bperf [1], and Intel VTune [2], which provides top-down analysis.

The profiling results provide two key insights. First, the sampling results obtained through bperf (Figure 1) revealed that the symmetric Gauss-Seidel (SYMGS) kernel, executed during the solving phase of the HPCG benchmark, is serialized. Optimizing this kernel has been identified as a key factor in improving the application's performance. Second, the top-down analysis using Intel VTune (Figure 2) revealed that the CPU pipeline slots are back-end bound, indicating the need for optimization in both memory access and computation instructions.

Based on the profiling results, we can devise the following optimization strategies. First, by leveraging Intel intrinsics, we can resolve the core-boundness of the SYMGS kernel's internal operations through SIMD (single instruction multiple data) parallelization. Additionally, memory prefetching can be employed to reduce stalls caused by cache misses when accessing the sparse matrix used in the SYMGS kernel's internal computations.

```
Samples: 65K of event 'task-clock', Event count (approx.): 65682000000
  Overhead  Command      Shared Object     Symbol
+   92.30%  xhpcg_perf   xhpcg_perf        [.] ComputeSYMGS_ref
+  [illegible]  xhpcg_perf   xhpcg_perf        [.] ComputeSPMV_ref
+  [illegible]  xhpcg_perf   libgomp.so.1      [.] gomp_team_barrier_wait_end
Samples: 9K of event 'task-clock', Event count (approx.): 54095000000
  Overhead  Command      Shared Object     Symbol
+   83.94%  xhpcg_perf   libgomp.so.1      [L] gomp_barrier_wait_end
+    9.93%  xhpcg_perf   libgomp.so.1      [.] gomp_barrier_wait_end
+  [illegible]  xhpcg_perf   xhpcg_perf        [.] ComputeSPMV_ref
+  [illegible]  xhpcg_perf   libgomp.so.1      [.] gomp_team_barrier_wait_end
```

**Figure 1 bperf sampling results. Main thread (above) and OpenMP threads (below).**

| | Pipeline Slot Type | | | | |
|---|---|---|---|---|---|
| | Retiring | Front-end bound | Back-end bound | | Bad speculation |
| | | | Core | Memory | |
| **Overhead (%)** | 10.4 | 0.7 | <u>43.6</u> | <u>45.2</u> | 0.1 |

**Figure 2 Top-down analysis results.**

**Acknowledgments** This work was supported in by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2023-00321688).

## References

[1] Ahn, Minwoo, et al. "Identifying On-/Off-CPU Bottlenecks Together with Blocked Samples." *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. 2024.

[2] Intel VTune. https://www.intel.com/content/www/us/en/docs/vtune-profiler/get-started-guide/2024-0/overview.html