DisCostiC: Digital Twin Performance Simulations Unlocking Hardware-Software Interplay

Ayesha Afzal, Georg Hager, Gerhard Wellein

Problem statement

MPI-parallel distributed-memory applications can exhibit unpredictable behavior when subjected to disturbances such as system noise, application noise, or imbalances. The typical "lock-step" pattern found in many numerical codes may break down, causing the program to enter a desynchronized state where computation naturally overlaps with communication. As a result, simply summing computation time and communication time may not accurately predict the total runtime. This raises important questions about the dynamics of memory-bound parallel applications and whether desynchronization, which facilitates bottleneck evasion and communication-computation overlap, is always beneficial for improving resource utilization. However, understanding this behavior on real systems is challenging due to ambiguous effects, such as system noise and variations in MPI implementations, and achieving a disturbance-free, controlled environment is not feasible. One solution to studying this complex hardware-software interaction is to use simulators, which provide controlled environments for experimentation. However, existing runtime simulators – such as SST, BigSim, xSim, LogGOPSim, Dimemas, and SimGrid – are trace-based and rely on traditional methods, often running directly on the target host architecture. These simulators don't adequately account for node-level bottlenecks and do not leverage analytical performance models to enable intuitive architectural exploration.

Our research approach and its strengths

We present an evaluation of the performance and dynamics of massively parallel MPI applications using our cross-architecture, full-scale parallel simulation framework. The key innovation of DisCostiC lies in its approach: rather than running code on actual hardware, it utilizes application skeletons built with a Domain-Specific Embedded Language (DSEL). Unlike trace-based simulators, this method accurately encodes inter-process dependencies without introducing ambiguous effects. The machine model encompasses the entire hierarchy of parallel systems – cores, chips, nodes, networks, and clusters – along with inherent bottlenecks such as memory bandwidth and the interactions between

these components. DisCostiC integrates the application model, machine model, performance models (including the Roofline model, Execution-Cache-Memory (ECM) model, Hockney's model, and LogGP variants), and the MPI implementation to generate simulated traces of the application. These traces can be visualized using tools like the Google Chromium web browser, ITAC, or Vampir.

Poster structure

In this poster, we first demonstrate the exploration of simulations within the framework, and then present experiments validating its efficiency and scalability.

Capabilities with two showcases

Hierarchical, hybrid, and heterogeneous (H3) clusters, such as Wisteria/BDEC-01, present additional challenges, including slow inter-cluster communication and varying network and memory bandwidth characteristics between clusters. Along with the Fujitsu MPI library, the WaitIO-MPI wrapper [1], built on Intel MPI, measures the network characteristics of MPI applications across heterogeneous clusters. Currently, DisCostiC uses the "socket" mode of WaitIO-MPI for inter-cluster communication via the InfiniBand interconnect, rather than the "file" mode (file system) or the "hybrid" mode (which selects either socket or file based on message size). We investigate how the performance of one cluster (Odyssey) affects the performance of another cluster (Aquarius) when both run a balanced workload. The experiment illustrates the dynamics of the memory-bound 2D four-point Jacobi code, using one A64FX node on Odyssey and one Ice Lake chip on Aquarius, over 50 iterations and a domain size of $20,000^2$. With a 128-byte eager limit in WaitIO-MPI, the rendezvous protocol is employed. In the balanced workload scenario, slow communication in the A64FX and idle waves in A64FX processes – rippling delay caused by slow Ice Lake processes and slow inter-cluster communication – result in fewer overlapping compute processes, ultimately improving the effective bandwidth per process through process desynchronization. In the second showcase, we conducted an experiment comparing the simulation and real-run performance of a memory-bound Jacobi application on 10 fixed nodes, with tasks per Ice Lake NUMA domain varying from 1 to 18 for 10,000 iterations on a $20,000^2$ domain. The results demonstrate that DisCostiC accurately simulates non-scaling behavior across cores within a single ccNUMA domain.

Accuracy, efficiency and scalability

We assess the accuracy of our simulation framework by comparing the performance of various proxy structures from real-world applications, such as Chebyshev filter diagonalization, Gauss-Seidel Successive Over-Relaxation (GSSOR), High-Performance Conjugate Gradients, and Optical Flow Solvers, across both Intel (Ice Lake ICL, Sapphire Rapids SPR) and non-Intel (Odyssey) systems.

Additionally, we replicated the experiments from the first showcase, modifying the setup to balance execution times on both Odyssey and Aquarius. To evaluate accuracy, we examine the error between actual and simulated runs, with error being a typical metric. We establish an acceptable error threshold to account for uncertainties, but also recognize the impact of ambiguous noise effects in real systems, which can influence the effectiveness of error metric. In our experiments across all strong and weak scaling cases, the error remained below 2%, with the errors marked as data point labels in the second graph. The error remained consistent across multiple iterations, indicating that the simulations closely align with the accuracy of the underlying performance models, further validating the reliability of our simulation framework for real-world performance prediction. To evaluate the simulator's performance, we ran twelve benchmarks and applications for approximately 1500 seconds on a single Sapphire Rapids node. For longer executions, the simulation time represented less than 1-2% of the total runtime, demonstrating the simulator's efficiency in scaling with the application's duration. However, for very short runs (less than one second), real executions were more efficient, as expected, due to the overhead of simulation. We also tested the simulator's scalability by simulating 4548 processes, consisting of 48 Odyssey processes and 4500 Aquarius processes. The simulation was executed with 4501 processes (simulation processes + 1) on Fritz hardware, which uses Ice Lake CPUs and an InfiniBand network. The results confirm the simulator's scalability, demonstrating its ability to handle large numbers of processes effectively.

Outlook and future work

The findings demonstrate that the simulator enables model-based design-space exploration, allowing the study of interactions between system components and the performance characteristics of complex parallel systems. Ongoing work includes adding support for energy predictions, additional performance bottlenecks (e.g., cache bottlenecks), and experimentation with both accelerated and non-accelerated programs.