A Benchmark of C++ Cluster-Computing Libraries: MASS C++, HPX, and TOPC

Ahmed Bera Pay

CSS 700 Autumn 2025 Term Report

University of Washington

12/10/2025

Project Committee:

Dr. Munehiro Fukuda, Committee Chair

Dr. David Socha, Committee Member

Dr. Brent Lagesse, Committee Member

Introduction

Distributed-memory parallel computation remains essential to high-performance computing (HPC), supporting large-scale scientific simulation, engineering design, data analytics, and AI workloads. As applications grow, developers must choose programming models that balance scalability with programmability. While MPI and OpenMP remain dominant, their explicit communication and synchronization requirements impose significant development overhead, motivating the use of higher-level distributed C++ runtimes.

This project examines three such runtimes of MASS C++, HPX, and PM2 which represent distinct execution paradigms. MASS C++ provides a place–agent abstraction and bulk-synchronous execution model for distributed arrays and structured communication. HPX implements an asynchronous many-task runtime aligned with modern C++ standards, using futures, lightweight threads, and active messages over a global address space. PM2 targets irregular and dynamic workloads through distributed multithreading, remote thread creation, and user-level thread migration. Together, these systems span synchronous, asynchronous, and irregular execution models within C++.

Other candidates were considered. TOP-C was excluded due to outdated documentation and limited modern applicability, and Cilk was retained only as a conceptual reference for work-stealing models. MASS C++, HPX, and PM2 thus provide a focused yet diverse basis for comparative study.

Standard benchmark suites such as NAS, PolyBench, Rodinia, and BOTS are useful for evaluating low-level kernels or heterogeneous systems, but they do not directly address questions of programmability or abstraction overhead in higher-level C++ runtimes. Work involving MASS C++, HPX, and PM2 typically highlights different aspects of each system, so a comparison based on a shared set of algorithms and data decompositions can offer additional perspective. This research therefore, aims to provide such a structured comparison within the scope of the selected benchmarks.

To address this gap, this research develops a benchmark suite spanning dense matrix multiplication, stencil/CFD kernels, graph motif search, graph message-passing simulations, and random-walk/agent-based models, implemented uniformly in MASS C++, HPX, and PM2 (with MPI/OpenMP baselines where appropriate). Identical algorithmic structures will enable controlled comparison of performance, scalability, programmability, and abstraction costs.

The Autumn 2025 phase focused on (1) validating benchmark relevance; (2) analyzing runtime execution models; (3) defining library-agnostic algorithmic blueprints; and (4) establishing a unified evaluation methodology for Winter–Spring 2026. These deliverables form the technical foundation for implementation and experimental analysis in subsequent quarters.

2 Related Work

2.1 Justification for Benchmark Selection

The benchmark suite reflects computational motifs that dominate HPC workloads and industrial simulation pipelines. Together, the selected patterns span regular and irregular computation, structured and unstructured communication, and heterogeneous memory-access behavior, properties essential for evaluating MASS C++, HPX, and PM2 under comparable algorithmic conditions.

2.1.1 CFD and Stencil Computation

Computational Fluid Dynamics (CFD) is a cornerstone of scientific and engineering computing, relying heavily on stencil computations to simulate physical phenomena (e.g. fluid flow and heat transfer). CFD is widely used in industry for product design and optimization in aerospace, automotive, energy, and manufacturing. For example, automakers use high-fidelity CFD simulations of aerodynamics and engine combustion to improve vehicle performance, which are computationally intensive due to the model complexity and fine mesh resolution [4]. HPC is essential to meet these demands, with supercomputers enabling CFD analyses that reduce costly physical prototyping (e.g. cutting wind tunnel tests by supplementing them with simulation) [5]. Industry reports show strong market growth for CFD software and services: the global CFD market is projected to triple from roughly $2.6 billion in 2025 to $7.8 billion by 2033 (11.6% CAGR) [6]. This growth is driven by broad adoption across aerospace, automotive, energy, and manufacturing, where CFD-based simulation is now critical for innovation and efficiency [4]. CFD's importance is also reflected in national HPC initiatives, for instance, the U.S. identified advanced CFD as a key use case requiring exascale computing to model full turbulence and complex flight conditions beyond what current systems can handle [5]. Overall, stencil-based CFD workloads exemplify the high-value scientific applications of HPC, with significant academic interest (for improving algorithms and fidelity) and industry demand (for faster, more accurate simulations that save time and cost in engineering design).

2.1.2 Graph Subgraph/Motif Search

Searching for subgraphs (patterns or motifs) within large graphs is a fundamental operation with growing importance in both social network analysis and biological network analysis. In social networks, subgraph matching is used to detect recurring community structures or communication patterns. For example, tracking how information cascades through a network or identifying cohesive groups of users [7]. The scale of modern networks (potentially billions of nodes) makes efficient subgraph search critical; indeed, it has become an active area of research in graph databases and streaming graph processing [8]. Industry applications abound: graph query engines can find complex fraud rings or supply-chain loops by spotting characteristic subgraph patterns in transactional and relationship data, a task at which graph databases excel compared to traditional SQL systems [9]. In the biological domain, subgraph search is equally vital. Biological networks (gene regulation networks, protein–protein interaction networks, metabolic pathways, etc.) are often analyzed by finding network motifs, which are small subgraphs that occur more frequently than chance. These motifs are recognized as "the simple building blocks of complex networks," helping researchers uncover functional circuit patterns in cells [10]. Similarly, in cheminformatics and drug discovery, finding subgraph matches underpins searching for molecular substructures. A recent study calls correlated subgraph search an "essential building block" for AI-powered drug discovery, as it allows researchers to identify candidate molecules in large chemical databases by matching substructure patterns [11]. In summary, academic interest in subgraph search is high (spanning graph theory, database optimization, and network science), and industry demand is rising with the proliferation of graph data in social media, finance (fraud detection), healthcare (molecule and disease networks), and beyond. Efficient subgraph search capabilities are increasingly viewed as crucial for extracting insights from big connected data.

2.1.3 Graph Message Passing

Graph-based message-passing algorithms are central to many agent-based simulations of markets and economies, where numerous agents (nodes) exchange information or "messages" along network links. In financial market simulations, for example, each trading agent's actions (bids, orders, trades) are communicated through a message-passing architecture that mimics real exchange protocols [12]. The ABIDES platform (Agent-Based Interactive Discrete Event Simulation), developed in academia and adopted by industry, employs a uniform message-passing system to coordinate thousands of trading agents in a simulated limit-order book market [12]. This approach has seen widespread use in the research community and at financial institutions (with support from firms like J.P. Morgan) because it can realistically model complex market dynamics with interacting algorithms. The demand for such simulations is driven by the need to test trading strategies, study systemic risks, and design market mechanisms in a controlled virtual environment. Commercial simulation engines like Simudyne illustrate industry's interest: Simudyne's agent-based market simulator enables stock exchanges and asset managers to analyze market and liquidity risks by simulating millions of agents in parallel on HPC infrastructure [13]. The software leverages distributed computing to scale these simulations, allowing researchers and analysts to run large-scale scenarios (e.g. entire equity markets or full-scale economies) efficiently. More broadly, graph message-passing algorithms underpin many distributed systems and graph analytics beyond finance – from information diffusion models to epidemic simulations – wherever entities on a network continuously update state based on neighbors' messages. The academic value is clear in fields like multi-agent systems and network science, and the market demand spans finance, economics, epidemiology, and defense (for modeling communication or contagion in networks). By serving as benchmarks, graph message-passing simulations ensure that computing platforms can handle the irregular communication patterns and heavy I/O of these complex, real-world workloads.

2.1.4 Dense Matrix Multiplication (DGEMM)

Matrix multiplication is one of the most ubiquitous and important computations in both scientific/engineering computing and modern AI. In numerical simulation and engineering analysis, large matrix operations arise in solving systems of equations, performing transforms, and implementing methods like finite element analysis. The general dense matrix–matrix multiply (GEMM) kernel has long been considered a "cornerstone" of high-performance computing due to its central role in scientific simulation workloads [14]. For instance, the LINPACK benchmark (used to rank the TOP500 supercomputers) measures how fast a system solves linear equations via matrix math, underscoring that HPC hardware is largely judged by its matrix throughput. Engineering software (from structural analysis to climate modeling) all rely on linear algebra libraries that are highly optimized for matrix-multiply performance. On the industry side, this importance is magnified by the rise of machine learning and AI. Matrix multiplication is the backbone of neural network computations: during both training and inference, layers of neural nets perform large matrix–vector and matrix–matrix multiplies. Consequently, accelerators like GPUs, TPUs, and specialized AI chips have been architected around fast GEMM operations. Academic literature notes that GEMM is "crucial to the acceleration of deep learning" and is the basic building block of the BLAS library used everywhere in scientific computing [14]. This dual role, enabling traditional HPC simulations and powering AI algorithms, means the demand for efficient matrix multiplication is at an all-time high. The HPC market (forecast to reach ~$87 billion by 2030 [19]) and the AI hardware market (projected to reach ~$76.7 billion by 2030 [20]) both invest heavily in better matrix-multiply performance. Researchers continue to publish improved algorithms

(e.g. recent breakthroughs in matrix multiplication theory and mixed-precision techniques [14]) and companies pour resources into software optimizations (like Intel's oneAPI Math Kernel Library or NVIDIA's cuBLAS) to maximize FLOPS. In short, matrix multiplication's value is universally recognized, and it is a well-rounded benchmark spanning scientific HPC, engineering applications, and commercial AI/ML, reflecting a core computational demand across domains.

2.1.5 Random Walk and Agent-Based Models

Random-walk based models refer to a class of agent-based or stochastic simulations often used to study complex behaviors in biological, ecological, or social systems. Despite their simple rules, these models are extremely important as they can generate realistic emergent phenomena, making them popular in both academic research and exploratory industry applications. A classic example is Wa-Tor, a predator–prey simulation proposed by A.K. Dewdney in 1984, where "sharks" and "fish" move on a grid (randomly or towards food) and reproduce or starve based on simple rules[15]. From those rules, complex population cycles emerge that resemble real ecological oscillations, illustrating the power of random agent behaviors to produce higher-level patterns. Such models help biologists and ecologists intuitively understand dynamics like predator–prey cycles or epidemic spread (when random walks represent movement of infected individuals). Another seminal model is Sugarscape, an agent-based social simulation introduced by Epstein & Axtell (1996) to study wealth distribution, trade, and cultural evolution. Sugarscape agents wander a grid seeking resources (sugar patches) and interacting (trading, reproducing, fighting) under simple behavioral rules. It is considered a "classic agent-based model" in social science, widely studied as a demonstration of how complex social phenomena (like inequality or migration patterns) can arise from simple individual behaviors[16]. In fact, Sugarscape is often cited as one of the first large-scale social simulations and is used in teaching and research as a benchmark for computational social science[16][17].

Beyond these famous examples, agent-based modeling (ABM) in general – which often relies on random-walk movement or probabilistic decision rules – has become an important tool in many domains. In economics and finance, ABMs with message-passing (as noted above) are used to simulate markets; in urban planning, they simulate how individuals move through transportation networks or how crowds behave; in epidemiology, random-walk models of people's contacts help predict disease outbreaks. The demand for ABM has grown with the need to analyze systemic risks and emergent outcomes that traditional equations can't capture. Industry and government agencies employ such simulations for scenario testing (e.g. pandemic responses or traffic flow optimization). This has led to efforts to run ever-larger ABMs: recent HPC implementations can simulate hundreds of millions of agents (each following random-based rules) in order to model entire economies or populations, executing in reasonable time on supercomputers[18]. The fact that ABM frameworks are scaling up to nation-sized simulations shows both their practical value and the computational challenge they pose. Random-walk based behavioral models cover a significant domain area – behavioral ecology, social dynamics, and complex systems science – making them a well-rounded benchmark category. They test a system's ability to handle many independent agents and stochastic events, and they provide insight into phenomena where aggregate behavior is not obvious from individual rules. Both academia and industry recognize the relevance of these models: academically, they are key to complexity science research, and in practice, they inform policy and strategy in environments as varied as financial markets, urban cities, and public health.

2.2 Differentiation From Prior Work

Existing comparative studies fall into three broad categories that leave gaps addressed by this project:

- **MPI/OpenMP-centric evaluations.** Suites such as NAS, PolyBench, Rodinia, and BOTS target kernel-level performance and hardware behavior rather than programmability or abstraction overhead, and do not evaluate modern distributed C++ runtimes as programming models.
- **Siloed evaluations of individual runtimes.** HPX studies emphasize FEM solvers, FFTs, and asynchronous scheduling; PM2 work focuses on irregular threading, migration, and load balancing; MASS C++ evaluations focus mainly on ABM scalability. None perform algorithmically equivalent comparisons across runtimes representing distinct execution paradigms.
- **Limited integration of programmability and performance.** Prior work rarely measures developer effort (e.g., LoC, boilerplate, synchronization complexity, debugging cost) alongside runtime performance using a consistent methodology.

2.3 Challenges Identified in Prior Work

Several methodological challenges highlighted in prior literature shape the benchmark design:

- **Semantic equivalence.** Runtimes differ in synchronization (bulk-synchronous vs. futures vs. migration), memory semantics, and communication patterns, making controlled comparison non-trivial.
- **Heterogeneous communication patterns.** Stencils exhibit structured neighbor exchange, while graph motifs, message passing, and ABMs require irregular, input-dependent communication. No runtime performs uniformly well across all pattern types, motivating a mixed benchmark set.
- **Measuring programmability.** There is no consensus on metrics for developer effort; LoC and modularity capture only part of cognitive and debugging complexity, so multi-faceted evaluation is needed.
- **Hardware sensitivity and reproducibility.** Distributed performance varies with interconnect topology, scheduling, and OS behavior, requiring transparent reporting of configurations and reproducible experimental procedures.
- **Abstraction vs. control.** High-level runtimes introduce overhead (e.g., task scheduling, barriers, migration) whose impact is workload-dependent, reinforcing the need for empirical, cross-model evaluation rather than purely theoretical comparison.

3 Implementation

3.1 Autumn 2025 Plan

Autumn 2025 established the methodological foundation required for semantically aligned, reproducible implementations of the benchmark suite across MASS C++, HPX, and PM2. Work proceeded through four coordinated tasks.

1. **Environment Setup and Runtime Familiarization.**

   A unified development environment was configured across local systems and the CSSMPI cluster, including installation of libraries and runtimes. Example programs were studied

to compare execution semantics, memory ownership rules, communication primitives, synchronization models, and runtime overheads.

2. **Benchmark Survey and Selection.**

   A literature survey of NAS, PolyBench, Rodinia, BOTS, and domain-specific HPC workloads guided the selection of five computational patterns: stencil/CFD, DGEMM, graph motif search, graph message passing, and random-walk/agent-based models (with a socio-economic ABM variant). Selection criteria included domain relevance, computational diversity, cross-runtime implementability, and alignment with contemporary HPC and simulation practice.

3. **Algorithmic Blueprints and Library-Agnostic Specifications.**

   Because MASS C++, HPX, and PM2 employ fundamentally different execution paradigms, bulk-synchronous, asynchronous many-task, and distributed multithreading with migration, the Autumn phase focused on defining runtime-neutral algorithmic specifications (see Appendix A and B). These blueprints including the basic core structures of the program are designed so that runtime performance differences will reflect runtime behavior rather than algorithmic variation.

4. **Evaluation Methodology and Integration Plan.**

   A unified evaluation framework was established for Winter–Spring 2026, including strong/weak scaling strategies, instrumentation for timing and communication profiling, load-balance analysis, programmability metrics (LoC, boilerplate ratio, modularity, synchronization complexity), and qualitative developer-effort logs. This plan aims for consistent, comparable experiments across runtimes.

3.2 Current Status

By the end of Autumn 2025, all preparatory objectives were completed, enabling full implementation work to begin in Winter 2026.

1. **Comparative Runtime Analysis**

   A study of each runtime clarified communication mechanisms (messages, futures, remote threads), synchronization semantics (barriers, continuations, migration triggers), memory abstractions, task/thread scheduling, and expected scaling characteristics for regular and irregular workloads. This informs how each benchmark blueprint maps to runtime-specific constructs.

2. **Benchmark Specifications**

   For all benchmark categories, formal specifications now include problem definitions, input/output assumptions, and runtime-independent execution semantics.

3. **Implementation Blueprints**

   Pseudo-code and mapping notes exist for each benchmark category. Each blueprint identifies how algorithmic components correspond to MASS places/agents, HPX lightweight threads and futures, and PM2 distributed threads, remote invocations, and migration policies. No implementation has yet been written, consistent with the planned Winter timeline.

4. **Evaluation Framework Ready.**

   Strong/weak scaling procedures, profiling instrumentation, reproducible cluster workflows, and visualization templates are prepared. Programmability metrics and analysis rubrics have also been defined. These preparations ensure that Winter implementation and Spring evaluation will proceed efficiently and systematically.

4 Evaluation

This section defines the evaluation methodology that will be applied once benchmark implementations are completed in Winter 2026. It specifies global performance metrics, benchmark-specific criteria, programmability measures, and a preliminary qualitative assessment based on the Autumn 2025 architectural analysis.

4.1 Evaluation Plan

The comparative study is organized along two dimensions:

- **Quantitative runtime performance**, and

- **Qualitative programmability and developer effort.**

  Qualitative programmability is analyzed in terms of the semantic gap between an algorithm's conceptual structure and its realization using runtime abstractions. This gap can reflect how directly algorithmic control flow maps to runtime constructs, how explicitly synchronization and communication must be expressed, and how much coordination logic is required beyond the core algorithm. Although these aspects are not directly quantifiable, they are examined through structured, workload-specific comparisons across runtimes to assess abstraction expressiveness and alignment with algorithmic intent.

  Each benchmark also includes workload-specific metrics reflecting its computational and communication characteristics.

4.1.1 Global Runtime Metrics

The following metrics will be collected across all benchmarks and runtimes:

- **Execution time:** wall-clock time of the benchmarked phase.
- **Strong scaling efficiency:** deviation from ideal speedup with fixed problem size.
  Weak scaling efficiency: runtime growth when problem size scales with node count.
- **Communication cost:** number of messages, mean message size, communication/computation ratio, synchronization frequency.
- **Load balance:** per-thread/node work variance, including HPX scheduling overhead and PM2 migration effectiveness.
- **Memory footprint:** peak distributed memory and allocation overhead.
- **Instrumentation:** Slurm timing, Linux perf, HPX performance counters, PM2 traces, MASS profiling hooks.

4.1.2 Benchmark-Specific Metrics

Each workload stresses a distinct slice of the runtime design space.

- **Stencil / CFD computation**

  Metrics include halo (ghost-cell) exchange cost, communication/computation ratio per iteration, barrier synchronization overhead, and sensitivity to block size and dimensionality. This benchmark probes the efficiency of neighbor communication and synchronization, stressing MASS's bulk-synchronous barriers, HPX's task granularity, and PM2's messaging under regular patterns.

- **Dense matrix multiplication (DGEMM).**

  Metrics include GFLOP/s throughput, sensitivity to tile/block size, memory bandwidth utilization, and overhead of distributed block ownership and synchronization. DGEMM provides a structured, communication-light baseline for comparing runtime overheads relative to optimized MPI/OpenMP implementations.

- **Graph subgraph/motif search.**

  Metrics include the number of remote edge traversals, work imbalance across nodes, latency sensitivity in pointer-chasing workloads, and performance sensitivity to graph degree distribution. This highlights irregular access and load imbalance, where HPX's fine-grained tasks, PM2's migration, and MASS's bulk-synchronous structure will likely behave differently.

- **Graph message-passing simulation.**

  Metrics include message throughput (messages/sec), latency distribution for small messages, update propagation time across the graph, contention under high message load, and per-agent update overhead. This benchmark models communication-driven simulations and evaluates PM2's remote-thread creation, HPX's active messages and futures, and MASS's collective synchronization.

- **Random-walk / agent-based models.**

  Metrics include agent update throughput, spatial load imbalance (hotspots), communication arising from local interactions, global synchronization overhead (if present), and local neighborhood contention. This workload mirrors socio-economic and ecological ABMs and probes MASS's place/agent mapping, PM2's dynamic load balancing, and HPX's tasking overhead for large agent populations.

4.1.3 Programmability and Developer Effort

This dimension evaluates the software engineering burden required to implement each benchmark in each runtime. Metrics include:

- **Lines of code (LoC):** for computation logic, communication code, and initialization.
- **Boilerplate ratio:** fraction of code required solely due to runtime API structure.
- **Synchronization complexity:** conceptual and structural burden of futures and continuations (HPX), remote thread creation and migration (PM2), and agent/place interactions and barriers (MASS).

- **Debugging overhead:** frequency and difficulty of runtime-specific errors observed during development.
- **Idiomatic fit:** how naturally each benchmark maps onto each runtime's abstractions.
- **Lack of Cohesion of Methods (LCOM):** class-level cohesion metrics are considered where applicable to assess structural cohesion in object-oriented components; however, their applicability and interpretability vary across runtimes due to differences in programming style, abstraction granularity, and use of procedural or task-based constructs.

4.2 Preliminary Qualitative Evaluation

Because implementations have not yet been executed, this assessment reflects architectural analysis performed during Autumn 2025.

4.2.1 Expected Runtime Characteristics

At a high level, the expected behavior is:

- **Stencil / CFD:** MASS C++ is expected to perform strongly due to structured bulk-synchronous phases; HPX performance will depend on grouping tasks to avoid fine-grained overhead; PM2 should be stable but may gain little from its dynamic features on regular workloads.

- **DGEMM:** All runtimes should scale well, with differences arising from overheads. HPX will require tuning of block size to match scheduling granularity, MASS may incur barrier costs for small tiles, and PM2 will incur messaging overhead for block exchanges.

- **Graph subgraph search:** HPX is expected to benefit from fine-grained asynchronous tasks; PM2's migration may mitigate imbalance in skewed graphs; MASS may face challenges expressing irregular communication without excessive synchronization.

- **Graph message passing:** PM2's communication model is well suited for high-volume small messages; HPX can perform well with appropriate grain size for active messages; MASS may introduce synchronization bottlenecks unless communication is carefully structured.

- **Random-walk / ABM:** MASS C++ is a natural fit due to its agent/place model; PM2 may perform well in dynamically imbalanced scenarios; HPX will likely require aggregation of agent tasks to control scheduling overhead.

4.2.2 Anticipated Bottlenecks

Anticipated bottlenecks include:

- **MASS C++:** global barrier overhead and difficulty expressing highly irregular communication.
- **HPX:** scheduling overhead for overly fine-grained tasks and increased complexity in coordination.

- **PM2:** migration cost and debugging complexity associated with distributed threading behavior.

4.2.3 Risks and Open Questions

Open questions to be addressed empirically in Winter and Spring 2026 include:

- HPX's multi-node performance under heavy task loads.
- PM2's scalability for large, highly irregular graphs.
- Overhead of mapping non-ABM workloads onto MASS abstractions.
- Sensitivity of performance to decomposition and partitioning parameters.
- Variability in communication latency and its impact on cross-runtime comparisons.

5. Conclusion and Winter 2026 Plan

5.1 Summary of Autumn 2025 Status

The Autumn 2025 phase established the methodological and architectural groundwork for a controlled comparison of MASS C++, HPX, and PM2. The completed work ensures that forthcoming implementations will be semantically aligned, reproducible, and directly comparable.

A detailed runtime model analysis clarified the execution semantics of MASS's bulk-synchronous agent/place design, HPX's asynchronous many-task runtime with futures and lightweight threads, and PM2's distributed multithreading with remote thread creation and migration. Their communication primitives, synchronization behavior, memory abstractions, and expected scaling characteristics were documented to guide equivalent algorithmic mappings.

Five benchmark categories —stencil/CFD kernels, DGEMM, graph motif search, graph message-passing simulation, and random-walk/ABM— were selected based on domain relevance, computational diversity, and ability to expose complementary runtime strengths. Together, they span regular and irregular computation and a range of communication and synchronization patterns.

A complete runtime-neutral specification was produced for all benchmarks, defining decomposition choices, communication and synchronization phases, update semantics, and distributed state ownership rules. These blueprints, summarized in an implementation plan (see Appendix A and B), are intended to ensure that performance differences reflect runtime behavior rather than algorithmic variation.

Finally, a unified evaluation framework was defined, covering global performance metrics (execution time, scaling, communication cost, load balance), benchmark-specific criteria, and programmability metrics (lines of code, boilerplate ratio, synchronization complexity, debugging effort, idiomatic fit). A preliminary qualitative assessment, based on runtime architectures, identified expected strengths, bottlenecks, and open questions.

5.2 Winter 2026 Plan

Winter 2026 transitions the project from design to implementation and early experimentation. Work will proceed through five phases:

1. **Benchmark implementation.**

   Benchmarks will be implemented on MASS C++, HPX, and PM2 in the following order: stencil/CFD, DGEMM, random-walk ABM, graph motif search, and graph message passing.

Implementations will adhere to the Autumn blueprints to preserve semantic equivalence.

2. **Validation and correctness testing.**
   Correctness will be verified using small deterministic inputs, cross-runtime output comparisons, and model-specific invariants. Cluster deployment scripts will also be validated on CSSMPI.

3. **Initial performance experiments.**
   Stencil and DGEMM implementations will serve as early test cases to validate scaling behavior, identify bottlenecks, verify instrumentation, and tune runtime parameters such as HPX grain size and PM2 migration settings.

4. **Refinement and optimization.**
   Profiling results may motivate adjustments to decomposition strategies, communication phases, or runtime-specific tuning parameters. All modifications will be documented to maintain transparency in later comparisons.

References

[1] MASS: A Parallelizing Library for Multi-Agent Spatial Simulation. Distributed Systems Laboratory, University of Washington. Accessed: Dec. 22, 2025. [Online]. Available: http://depts.washington.edu/dslab/MASS/

[2] H. Kaiser, T. Heller, B. Adelstein-Lelbach, A. Serio, and D. Fey, "HPX: A Task Based Programming Model in a Global Address Space," *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models (PGAS '14)*, ACM, 2014, Article 6, pp. 1–11. https://doi.org/10.1145/2676870.2676883

[3] F. Namyst and P. Pierre, "PM2: A Distributed Multithreaded Programming Environment," *Proceedings of the 5th International Workshop on Parallel Programming Models*, 1996

[4] Rescale, "Automotive Simulation Solutions (HPC): CFD/FEA in Automotive Design." [Online]. Available: https://rescale.com/solutions/by-industry/automotive/

[5] *The Register*, "Obama endorses 3D TLC flash. How else can you do exaflop computing?" (NSCI/exascale discussion including turbulence-resolved CFD). [Online]. Available: https://www.theregister.com/2015/07/30/exaflop_supercomputing_and_exabyte_storage/

[6] Times EV, "Computational Fluid Dynamics Market Accelerates with AI, Digital Twins and HPC Integration," 2025. [Online]. Available: https://www.timesev.com/computational-fluid-dynamics-market-accelerates-with-ai-digital-twins-and-hpc-integration/

[7] "Accelerating Streaming Subgraph Matching via Vector Databases," *Science Partner Journal iComputing*. [Online]. Available: https://spj.science.org/doi/10.34133/icomputing.0131

[8] "Mnemonic: A Parallel Subgraph Matching System for Streaming Graphs," *IEEE IPDPS*, 2022. [Online]. Available: https://par.nsf.gov/servlets/purl/10377943

[9] Dataversity, "The Power of Graph Databases to Detect Fraud," 2023. [Online]. Available: https://www.dataversity.net/articles/the-power-of-graph-databases-to-detect-fraud/

[10] U. Alon, “Biological Network Motif Detection: Principles and Practice,” *Briefings in Bioinformatics*, 2012. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/22396487/

[11] “Efficient Correlated Subgraph Searches for AI-powered Drug Discovery,” *IJCAI*, 2024. [Online]. Available: https://www.ijcai.org/proceedings/2024/0260.pdf

[12] “A Financial Market Simulation Environment for Trading Agents Using Deep Reinforcement Learning,” *ACM ICAIF*, 2024. [Online]. Available: https://strategicreasoning.org/wp-content/uploads/2024/11/ICAIF24proceedings_PyMarketSim.pdf

[13] AWS HPC Blog, “Harnessing the Power of Agent-based Modeling for Equity Market Simulation and Strategy Testing,” 2024. [Online]. Available: https://aws.amazon.com/blogs/hpc/harnessing-the-power-of-agent-based-modeling-for-equity-market-simulation-and-strategy-testing/

[14] “The Cambrian Explosion of Mixed-Precision Matrix Multiplication for Quantized Deep Learning Inference,” *arXiv*, 2025. [Online]. Available: https://arxiv.org/html/2506.11728v1

[15] “Wa-Tor,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Wa-Tor

[16] “Sugarscape Replication Study,” George Mason University (CS), 2007. [Online]. Available: https://cs.gmu.edu/~sean/papers/Replication07.pdf

[17] “Sugarscape,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Sugarscape

[18] “High-performance Computing Implementations of Agent-based Economic Models for Realizing 1:1 Scale Simulations of Large Economies,” *IEEE Transactions on Parallel and Distributed Systems*, 2021. [Online]. Available: https://pure.iiasa.ac.at/id/eprint/17104/

[19] Grand View Research (2025). High Performance Computing Market (2025–2030): Size, Share & Trends Analysis Report. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/high-performance-computing-market

[20] ResearchAndMarkets.com, “Artificial Intelligence (AI) Hardware Market — Types, Applications and Industry Sectors: 2021-2030 Forecast.” Available: https://www.globenewswire.com/news-release/2025/11/06/3182808/0/en/Artificial-Intelligence-Hardware-Market-Analysis-and-Forecast-2021-2030-Types-Applications-Industry-Sectors-and-the-Competitive-Landscape.html

Appendix A — Benchmark Specifications

This appendix summarizes the formal benchmark definitions used in the comparative evaluation of MASS C++, HPX, and PM2. Each specification is runtime-neutral and focuses on problem definition, decomposition, communication structure, and correctness criteria.

A.1 3D Stencil / CFD Kernel (Heat3D)

Problem Description

Numerical integration of the 3D heat equation using a 7-point stencil. Each grid point updates based on its six axial neighbors and itself.

Domain

A uniform 3D mesh of size N_x × N_y × N_z.

Update Rule (7-point stencil)

For temperature field u(x,y,z,t):

$$u_{i,j,k}^{t+1} = \alpha u_{i,j,k}^{t} + \beta\left(u_{i-1,j,k}^{t} + u_{i+1,j,k}^{t} + u_{i,j-1,k}^{t} + u_{i,j+1,k}^{t} + u_{i,j,k-1}^{t} + u_{i,j,k+1}^{t}\right)$$

Parameters α, β chosen to satisfy stability constraints.

Decomposition

- 3D block partitioning across P_x × P_y × P_z.
- Each block includes a **halo layer of width 1** on all faces.

Required Communication per Iteration

- 6 face exchanges (±x, ±y, ±z) per block.
- Optional edge/corner exchange if needed by implementation (not required by a 7-point stencil).

Correctness Criteria

- Matching numerical output with an MPI reference implementation for fixed timesteps.
- Preservation of total energy under periodic or closed boundary conditions.

A.2 Dense Matrix Multiplication (DGEMM)

Problem

Compute C = A × B for dense double-precision matrices.

Decomposition

- Matrices are partitioned into b × b tiles.
- Tiles assigned to a logical 2D process grid.
- SUMMA-style communication: broadcast needed A blocks along rows and B blocks along columns.

Communication Structure

- Row broadcasts of A-tiles.
- Column broadcasts of B-tiles.
- No additional synchronization except per-panel progress.

Correctness Criteria

- Bitwise or tolerance-based equivalence to reference BLAS implementation.

A.3 Graph Subgraph / Motif Search

Problem

Count or enumerate occurrences of a small pattern graph H (size 3–5) inside a large input graph G.

Decomposition

- Vertex partitioning with ghost vertices for boundary edges.
- Local work begins from vertices owned by each process.

Communication Structure

- Expand partial matches requiring remote vertices.
- Exchange of frontier vertices or tasks, depending on runtime.
- Optional aggregation of counts before global reduction.

Correctness Criteria

- Total motif count must match a serial backtracking baseline.

A.4 Graph-Based Message-Passing Simulation

Problem

At each timestep, each vertex sends a message to its neighbors and updates its state based on received messages.

Decomposition

- Each process stores a subset of vertices and incident edges.
- Edges crossing partitions cause message traffic.

Communication Structure

Per timestep:

- Outbound messages to neighbors (local or remote).
- Delivery phase ensuring all vertices have received messages for step t.
- Local state update.

Correctness Criteria

- Deterministic consistency across all runtimes for fixed topology and update rules.

A.5 Random-Walk / Agent-Based Model (ABM)

Problem

Simulate large numbers of agents performing random walks (and optional interactions) on a 2D grid.

Decomposition

- Spatial domain split into blocks.
- Agents transferred across partitions as they cross boundaries.

Communication Structure

- Migration of agents between owning partitions.
- Optional message exchange for agent-agent interactions.

Correctness Criteria

- Preservation of agent counts, valid movement rules, and matching distributions with baseline runs.

Appendix B — Dataset and Parameter Tables

This appendix defines representative problem sizes, including small (debug), medium (development), and large (cluster-scale) configurations. The dataset sizes and parameter ranges listed here represent the planned initial configurations for the Winter 2026 implementations. These values are provisional and may be adjusted during development after early correctness and scaling tests.

B.1 3D Stencil / CFD Benchmark (Heat3D)

Table B.1 — Parameters

Parameter	Symbol	Example Values	Notes
Grid size	N_x,N_y,N_z	128^3, 256^3, 512^3, 1024^3	3D domain
Timesteps	T	50, 200, 500	Simulation duration
Coefficients	α,β	e.g., 0.5, 1/12	Stable for explicit heat equation
Decomposition	P_x,P_y,P_z	1×1×1 → 4×4×4	Matches node/core grid
Halo width	—	1	7-point stencil

Size tiers

Tier	Grid Size	Timesteps	Use Case
Small	128 × 128 × 128	50	Debug correctness
Medium	256 × 256 × 256	200	Scaling on few nodes
Large	512–1024^3	200–500	Cluster-scale evaluation

B.2 DGEMM Benchmark

Table B.2 — Parameters

Parameter	Symbol	Example Values	Notes
Matrix dimension	M,N,K	2048, 4096, 8192	Square matrices
Block size	b	64, 128, 256	Cache / runtime tuning
Process grid	P_x,P_y	1×1 → 4×4	SUMMA-style mapping
Data type	—	double	DGEMM specification

Size tiers

Tier	Matrix Size	Block Size	Notes
Small	2048	64	Debug / unit testing

Medium	4096	128	Mid-scale development runs
Large	8192	128–256	Stress memory + communication

B.3 Graph Subgraph / Motif Search

Table B.3 — Parameters

Parameter	Symbol	Example Values	Notes
Vertices	$\|V\|$	10^5, 10^6	Large sparse graphs
Edges	$\|E\|$	10^6–10^7	Power-law or ER
Motif size	k	3, 4, 5	Triangles, 4-cycles, etc.
Partitions	P	4, 8, 16	Vertex-cut partitioning

Size tiers

Tier	$\|V\|$	$\|E\|$	Motif
Small	10^5	10^6	k = 3
Medium	5×10^5	5×10^6	k = 3–4

Large	10^6	10^7–5×10^7	k = 4–5

B.4 Graph Message-Passing Benchmark

Table B.4 — Parameters

Parameter	Symbol	Example Values	Notes
Agents / nodes	|V|	10^4, 10^5, 10^6	Each vertex hosts an agent
Average degree	d	4, 8, 16	Controls message density
Timesteps	T	100, 500	Simulation length
Message size	—	8–64 bytes	Small status/control messages
Topology	—	grid, random, scale-free	Varies communication behavior

Size tiers

Tier	Agents	Degree	Timesteps
Small	10^4	4	100
Medium	10^5	8	200–500

Large	10^6	8–16	500

B.5 Random Walk / ABM Benchmark

Table B.5 — Parameters

Parameter	Symbol	Example Values	Notes
Grid size	L_x,L_y	512^2, 1024^2	2D spatial domain
Agents	A	10^4, 10^5, 10^6	Population size
Timesteps	T	500, 1000	Simulation duration
Movement model	—	von Neumann / Moore	4- or 8-neighborhood
Interactions	—	optional rules	Enables socio-economic ABMs

Size tiers

Tier	Grid Size	Agents	Timesteps
Small	512^2	10^4	500
Medium	1024^2	10^5	1000
Large	1024^2	10^6	1000+

Appendix C — Additional Sources Consulted

These sources informed background understanding and contextual framing but were not cited directly in the report.

[A1] Neo4j Blog. *Graph Database Use Cases*.

https://neo4j.com/blog/graph-database/graph-database-use-cases/

[A2] TechTarget. *Top 5 Enterprise Graph Analytics Use Cases*.

https://www.techtarget.com/searchbusinessanalytics/feature/Top-5-enterprise-graph-analytics-use-cases

[A3] Grand View Research. *Graph Database Market Report*.

https://www.grandviewresearch.com/industry-analysis/graph-database-market-report

[A4] TechTarget News. *Gartner Predicts Exponential Growth of Graph Technology*.

https://www.techtarget.com/searchbusinessanalytics/news/252507769/Gartner-predicts-exponential-growth-of-graph-technology

[A5] ACM Computing Surveys. *Graph Neural Networks: Foundations, Frontiers, and Applications*.

https://dl.acm.org/doi/10.1145/3725323

[A6] ACM Transactions / SIGMOD Blog. *Recent Trends in Graph Analytics* (Full-text article).

https://dl.acm.org/doi/full/10.1145/3732778

[A7] Oxford Academic. *Network Motifs: Simple Building Blocks of Complex Networks*.

https://academic.oup.com/bib/article/13/2/202/253539

[A8] UW DSLab. *MASS C++ Case Study (Panther, 2020)*.

https://depts.washington.edu/dslab/MASS/reports/SarahPanther_wi20.pdf

[A9] Runestone Academy. *Agent-Based Models — Sugarscape Tutorial*.

https://runestone.academy/ns/books/published/complex/AgentBasedModels/Sugarscape.html