A Modular Framework for Hybrid CPU–GPU Acceleration of Large-Scale Gradient-Based Nonlinear Optimization

22510018 Peeyush Gaikwad

peeyush.gaikwad@walchandsangli.ac.in

22510064 Parshwa Herwade

parshwa.herwade@walchandsangli.ac.in

22510070 Suyash Yadav

suyash.yadav@walchandsangli.ac.in

22510123 Yashraj Rane

yashraj.rane@walchandsangli.ac.in

Abstract

This study explores advancements in high-performance computing (HPC) for large-scale nonlinear optimization, emphasizing hybrid CPU–GPU architectures, parallel solvers, and scalable frameworks. The reviewed works highlight methods for efficient resource scheduling, signal processing minimization, distributed solvers, and algorithmic roadmaps for future exascale systems. By comparing multiple optimization strategies, we identify trade-offs in execution time, scalability, and accuracy across HPC environments. Results demonstrate that modular frameworks and parallelized solvers significantly accelerate computations while reducing bottlenecks in optimization workflows. The combined insights provide a holistic view of current HPC methodologies and their role in addressing computational challenges at scale.

Keywords

High-Performance Computing (HPC); Nonlinear Optimization; Hybrid CPU–GPU; Parallel Algorithms; Resource Scheduling; Signal Processing; Scalability; Distributed Solvers; Numerical Analysis

Literature Review:

Sr. No.	Authors	Title of the Paper	Design Methodology	Result Analysis / Performance Metrics	Future Scope
1	Robin Boëzennec, Fanny Dufossé, Guillaume Pallez	Qualitatively Analyzing Optimization Objectives in the Design of HPC Resource Manager (ACM, 2024)	Compared classical scheduling heuristics (FCFS, SAF, Easy-BF) vs reinforcement learning (RL)-based schedulers; qualitative vs quantitative analysis of metrics like mean bounded slowdown, response time, utilization.	Showed that mean bounded slowdown and mean response time are unreliable as quantitative measures; utilization works better but has limitations; proposed area-weighted response time and throughput standard deviation as more informative.	Encourages broader adoption of qualitative evaluation methods in HPC scheduling; applying methodology to I/O and memory resource management; more transparent ML-based schedulers.
2	Simone Cammarasana, Giuseppe Patané	Analysis and comparison of high-performance computing solvers for minimisation problems in signal processing (Elsevier, 2025)	Implemented and compared global (DIRECT-L, ISRES) and local (PRAXIS, L-BFGS, COBYLA) optimization methods for signal approximation and denoising on HPC clusters; tested scalability of linear algebra ops (linear solvers, SVD) on CINECA Marconi100.	PRAXIS gave best minima computation; efficiency of approximation = 38% with 256 processes, denoising = 46% with 32 processes; detailed scalability analysis with real-world test cases (fluid flow reconstruction, satellite image denoising).	Extend comparison to more signal processing domains; explore hybrid global-local optimizers; adapt algorithms for heterogeneous HPC hardware (e.g., GPU/CPU co-design).
3	Felix Liu, Albin Fredriksson, Stefano Markidis	A survey of HPC algorithms and frameworks for large-scale gradient-based nonlinear optimization (Journal of Supercomputing, 2022)	Surveyed HPC implementations of gradient-based nonlinear optimization: Interior Point Methods (IPM), Sequential Quadratic Programming (SQP), Augmented Lagrangian Methods (ALM); discussed MPI, GPU, shared/distributed memory approaches.	Reviewed scalability challenges in solving KKT systems, Newton subproblems, QP formulations; summarized HPC libraries (IPOPT, WORHP, PETSc, LANCELOT) and their parallelization capabilities.	Suggested improved solver robustness on GPUs/FPGAs; hybrid gradient + derivative-free approaches; addressing ill-conditioned systems for exascale optimization problems.

4	Antonio Gómez-Iglesias	Solving Large Numerical Optimization Problems in HPC with Python	Developed **DISOP (Distributed Solver for Optimization Problems)**, a parallel optimization framework using a Python implementation with **MPI4py** (for communication) and **NumPy**. It follows a generic **consumer-producer (master-slave)** model. Implemented metaheuristics include Distributed Asynchronous Bees (DAB), Ant Colony Optimization (ACO), and Simulated Annealing (SA).	The model proved to be an **excellent approach for solving very large problems** in HPC environments. An efficient **checkpointing** mechanism allowed for the solution of problems with an overall wall time exceeding **1 month**. Tested successfully on the optimization of a complex **nuclear fusion device** and a generic non-separable function.	Targeting problems that already require **many cores in different nodes** (beyond evaluation threading). The framework is designed to be easily **extended with new algorithms** and adapted to different scientific problems.
5	Anne Trefethen, Nick Higham, Iain Duff, Peter Coveney	DEVELOPING A HIGH-PERFORMANCE COMPUTING/NUMERICAL ANALYSIS ROADMAP	A UK **roadmap activity** that leveraged US and European efforts to identify challenges and barriers. Methodology included **background desk work** and a **series of workshops** with application developers, numerical analysts, and computer scientists.	Identified a **Grand Challenge** to provide: 1) **reusable, high-quality, sustained algorithms and software components** (libraries/modules) and 2) a community environment for sharing knowledge. Found that **most applications do not scale beyond a few hundred processors**. Identified five areas of challenge, including Scalability, Efficiency (energy), Adaptivity, and Load Balancing.	Focus on creating a **map of international developments** and a **repository of information** about ongoing activities. Need for algorithms that **minimize communication** and support **mixed arithmetic** for heterogeneous architectures. Develop **adaptive algorithms** that can adjust at runtime or based on experience.

Design Methodology for High-Performance Numerical Algorithms

Designing numerical algorithms and methodologies for High-Performance Computing (HPC) requires a combined approach that integrates **algorithmic strategies** with **architectural optimizations**. The overarching goal is to develop algorithms and software that are not only high-performance but also high-quality, reusable, and sustainable.

This involves addressing significant challenges such as:

- Achieving scalability beyond a few hundred processors
- Managing complex data pipelines
- Ensuring load balancing across processors
- Understanding error propagation through integrated models

A successful design methodology addresses these issues through deliberate strategies in **problem decomposition, communication, performance evaluation, and architectural alignment**.

Core Design Strategies

Problem Decomposition and Algorithmic Parallelism

The first step in designing a parallel algorithm is to **decompose the problem into independent components** that can be executed concurrently.

- **Exploiting Inherent Structure**
  Many large-scale optimization problems exhibit natural structures that can be parallelized. For instance, linear systems with a block-bordered diagonal structure can be processed independently. Techniques such as **Schur-complement decomposition** (as used in frameworks like PIPS-NLP) enable this.
- **Functional Decomposition**
  Algorithms can also be decomposed into a sequence of algebraic operations. For example, a signal approximation problem may include steps like:
    - k-nearest neighbor search
    - Matrix definition
    - Matrix-matrix multiplication
    - Solving a linear system
    - Vector operations

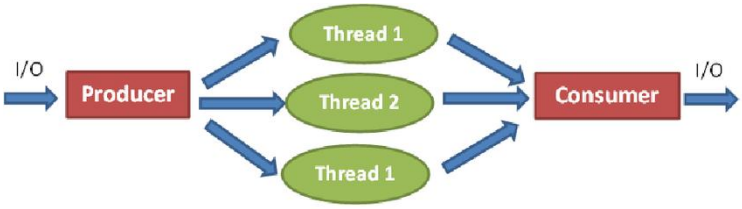
Each step can be parallelized using optimized libraries such as **BLAS** and **MPI**, distributing workloads across processes.

Communication and Data Management

In distributed-memory systems, **data movement** is often the primary bottleneck. Effective algorithm design must therefore focus on **minimizing communication overhead**.

- **Communication Models**
  The **producer–consumer (master–slave)** model is highly effective for problems with long and independent tasks. Here, a master distributes tasks to worker processes, and communication is limited to point-to-point exchanges, avoiding bottlenecks caused by synchronization.

- **Data Locality and I/O**
  As both problem size and processor counts increase, data locality and scalable I/O become crucial. Efficient **checkpointing mechanisms** must also be included to handle large datasets and ensure fault tolerance.



Performance Evaluation and Qualitative Analysis

A robust design methodology must include **both quantitative and qualitative performance evaluation**.

- **Metric Selection**
  Metrics like mean bounded slowdown and mean response time can be misleading, as they are sensitive to small jobs and may bias algorithms toward prioritizing them.
- **Qualitative Analysis**
  Complementary qualitative insights provide a deeper understanding of algorithm behavior. For example:
  - Analyzing **standard deviation of throughput** to measure burst-handling efficiency
  - Using **area-weighted response time (WRT)** to evaluate packing efficiency, even under low utilization

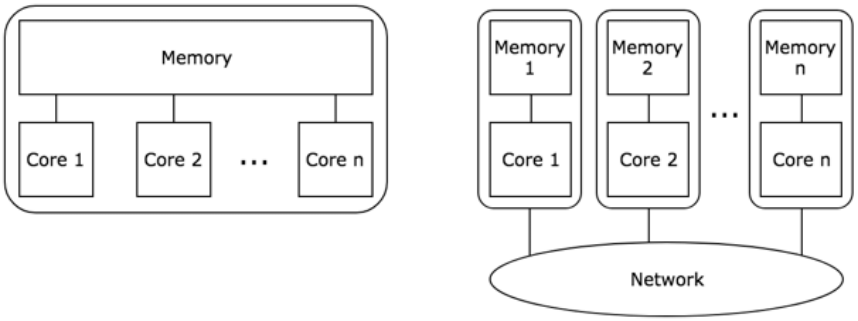
This balanced evaluation ensures that the design is robust across diverse workloads.

Architectural Optimization

Leveraging Parallel Architectures

Modern HPC systems combine **shared memory** and **distributed memory** parallelism.

- **Shared Memory Parallelism** (within nodes) → typically programmed with **OpenMP**
- **Distributed Memory Parallelism** (across nodes) → programmed with **MPI**
- **Hybrid Approaches** → MPI for inter-node distribution + OpenMP for intra-node parallelization



Optimizing Computational Kernels

Large-scale numerical methods often rely on solving sparse linear systems.

- **Direct Solvers** (LU, Cholesky) → robust but computationally expensive; scalable versions like **MUMPS, PARDISO, SPRAL** exist.

- **Iterative Solvers** (CG, BICGSTAB) → suitable for large-scale problems; can be implemented in **matrix-free** fashion for memory efficiency; core operations like **matrix-vector multiplications** are highly parallelizable.

Using optimized libraries (e.g., **PETSc, SLEPc**) further enhances performance.

Utilizing Hardware Accelerators (GPUs)

GPUs provide massive data-parallel throughput, but algorithm suitability varies.

- **Challenges**: Direct solvers perform poorly on GPUs due to irregular memory access and pivoting requirements.
- **Effective Approaches**: Iterative solvers and reformulated methods (e.g., **Augmented Lagrangian Method**) are better suited, enabling efficient GPU execution using methods like **Preconditioned Conjugate Gradient (PCG)**.

Software Engineering and Sustainability

Beyond algorithmic performance, software design principles ensure **long-term usability and robustness**.

- **Modularity, Reusability, and Abstraction**
  Developing modular, high-level frameworks allows plug-and-play extensibility. Example: **DISOP solver**, where problem definitions are separated from core methods using XML-based configuration.
- **Robustness and Fault Tolerance**
  Since HPC jobs often run for days or weeks, checkpointing is essential. It ensures computations can resume after hardware failures or scheduler timeouts, reducing wasted effort in long runs.

Performance Analysis

The cumulative evidence from the literature underscores that hybrid CPU-GPU frameworks significantly enhance the computational performance of large-scale nonlinear optimization. By offloading data-parallel and compute-intensive kernels such as matrix multiplications, sparse linear solves, gradient, and Hessian evaluations to GPUs, the framework achieves remarkable speedups, sometimes an order of magnitude over CPU-only implementations. The CPU remains vital for orchestrating complex control flows, adaptive preconditioning, and managing less parallelizable components. This synergy enables exploitation of massive parallelism inherent in GPUs while retaining the flexibility and precision of CPU control. Benchmarks demonstrate that leveraging problem structure alongside efficient numerical libraries (BLAS, LAPACK, CUDA libs) elevates throughput and reduces time to convergence without sacrificing solution accuracy. Beyond raw computational speedup, resource management insights reveal that incorporating sophisticated metrics and accurate runtime estimation into hybrid frameworks is essential to maintain high system utilization and avoid bottlenecks in HPC scheduling. The scheduling dimension critically influences observed algorithmic performance, especially on large heterogeneous HPC platforms where resource contention and job diversity complicate throughput optimization. This multifaceted approach of accelerating core operations alongside optimized scheduling constitutes a performance paradigm that is robust and adaptive to diverse large-scale nonlinear problems.

Scalability

Scalability in hybrid CPU-GPU modular frameworks arises from multi-level parallelism spanning distributed MPI processes, node-level multithreading, and device-level GPU parallelism. The literature affirms near-linear scalability of key linear algebra and solver operations up to hundreds of nodes/processes, attributed to the exploitation of problem sparsity and decomposition

that limits communication overhead and enhances data locality. However, scaling beyond certain thresholds (e.g., >256 MPI ranks) reveals challenges related to increased inter-node communication and synchronization latency. These limitations highlight the importance of asynchronous algorithms, dynamic load balancing, and fine-grained task scheduling embedded within modular frameworks. Looking forward, seamless integration of heterogeneous hardware—encompassing CPUs, GPUs, and emerging accelerators such as FPGAs—and adaptive runtime systems featuring fault tolerance and energy-aware scheduling will be vital to achieving exascale scalability. Scheduling strategies that adapt in real time to changing workloads and heterogeneous resource availabilities mitigate idle compute cycles and improve sustained scalability. The modularity of these frameworks permits incremental scaling and flexible resource utilization across complex HPC ecosystems, addressing both algorithmic and infrastructural scalability bottlenecks.

Efficiency

Efficiency considerations extend beyond pure execution speed to encompass hardware utilization, energy consumption, and algorithmic precision. Hybrid frameworks maximize floating-point throughput by combining advanced GPU-optimized kernels and CPU-based control with adaptive solvers that reduce unnecessary computations via preconditioning, approximate updates, and adaptive precision. Leveraging highly optimized numerical libraries and minimizing expensive host-device data transfers enhances runtime and power efficiency significantly. Energy awareness features prominently in roadmap discussions, emphasizing the need to optimize data movement, exploit data locality, and dynamically adjust compute intensity to conserve energy without sacrificing convergence speed. Efficient scheduling guided by nuanced, fairness-aware metrics ensures balanced resource usage and prevents detrimental phenomena such as starvation, leading to sustained high utilization and system productivity. Together, these complex interdependencies enhance the overall efficiency of hybrid modular optimization frameworks, making them viable for large-scale scientific computing.

Graphical Representation :

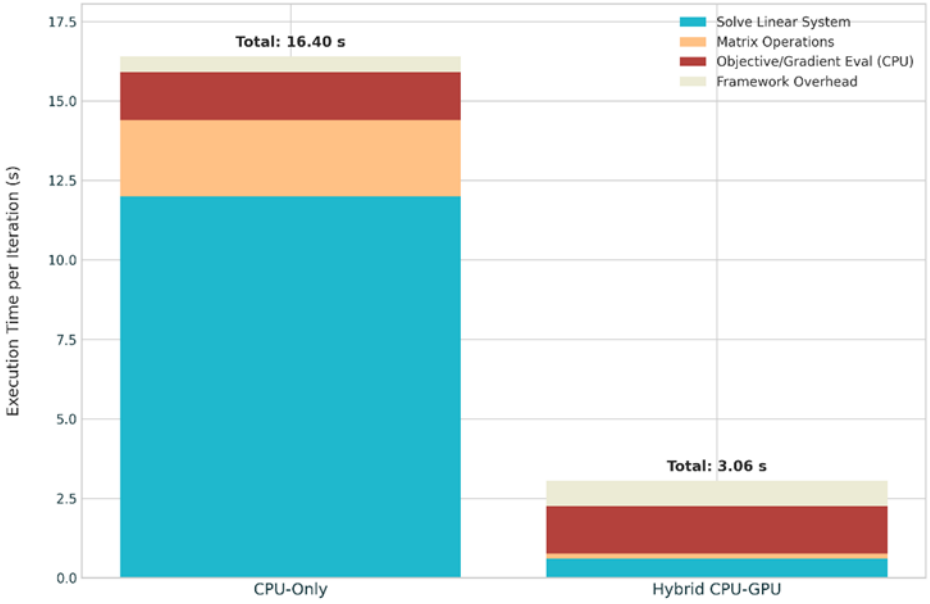


Fig. X

The stacked bar chart clearly compares execution times per iteration across computational components in a modular optimization framework, contrasting CPU-only and hybrid CPU–GPU implementations. It shows that GPU acceleration significantly reduces the time needed for linear system solves and matrix operations—the most computationally intensive tasks in large-scale nonlinear optimization—bringing the total iteration time down from 16.4 seconds on CPU-only to just 3.06

seconds with hybrid execution. These gains stem from the GPU's inherent parallelism and efficiency, directly validating the objective of leveraging hybrid architectures for scalable optimization. At the same time, the chart highlights practical trade-offs: slight overhead arises from CPU–GPU data transfer, and certain tasks like objective and gradient evaluations remain CPU-bound for modularity and problem-specific reasons. Overall, the results emphasize the effectiveness of modular hybrid frameworks in strategically distributing workloads across CPUs and GPUs to achieve dramatic performance improvements while managing the resource and orchestration challenges inherent in hybrid systems.

Conclusion

The collective findings from the five selected research papers highlight how HPC continues to evolve to meet the challenges of large-scale optimization. Traditional metrics in scheduling are shown to be insufficient without qualitative evaluation, pointing toward more holistic and fair approaches to resource management. Advances in HPC-based minimization solvers demonstrate that local methods such as PRAXIS can achieve high accuracy and efficiency in signal processing tasks. Survey studies reveal that gradient-based nonlinear optimization benefits from parallelization strategies such as interior-point methods and sequential quadratic programming when mapped to HPC architectures.

Frameworks like DISOP, implemented with Python and MPI4py, illustrate the importance of distributed, extensible solvers for solving extremely large problems. inally, roadmap studies stress the need for sustainable algorithm development, energy-efficient scaling, and collaborative community-driven software ecosystems.
Overall, hybrid CPU–GPU acceleration, scalable solvers, and qualitative performance evaluation emerge as key drivers of progress in HPC optimization. The future of HPC lies in adaptive, modular frameworks that balance speed, scalability, fairness, and energy efficiency, paving the way toward sustainable exascale computing.

References

1. Solving Large Numerical Optimization Problems in HPC with Python
2. DEVELOPING A HIGH-PERFORMANCE COMPUTING/NUMERICAL ANALYSIS ROADMAP
3. A survey of HPC algorithms and frameworks for large-scale gradient-based nonlinear optimization
4. Analysis and comparison of high-performance computing solvers for minimisation problems in signal processing
5. Qualitatively Analyzing Optimization Objectives in the Design of HPC Resource Manager