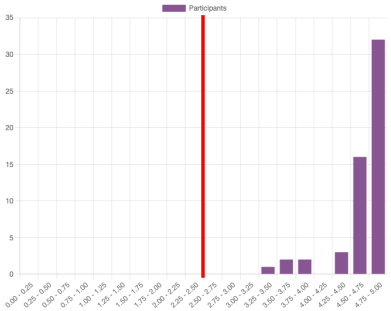
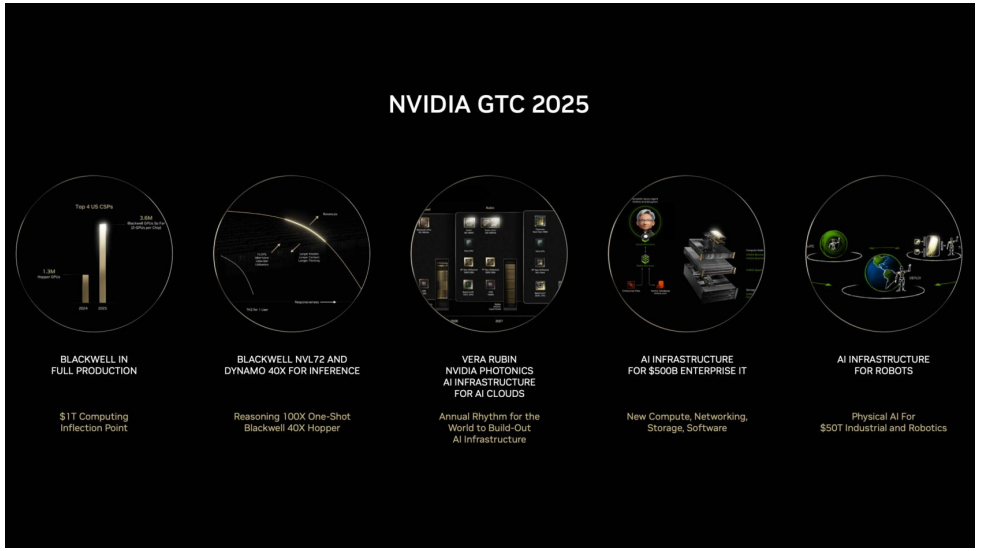
COMP4300 – Course Update

- **Quiz 1 results**



- **Assignment 1 is due 14 April, 11:55PM**
  - A well-written, solid explanation of the methodology and rationale pursued to solve the different tasks and the results obtained in your report is essential to pass the assignment

COMP4300 – News - Nvidia GTC 2025



COMP4300 – News - Nvidia GTC 2025

Bill Dally, chief scientist at NVIDIA, and Yann LeCun, chief AI scientist at Meta and a professor at New York University

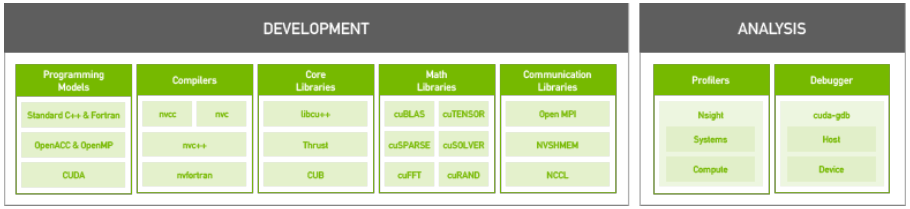
LeCun predicted that artificial general intelligence, or AGI — which he prefers to call advanced machine intelligence, because "human intelligence is superspecialized, so calling it general is a misnomer" — will be viable in three to five years.

LeCun also spoke about his work at Meta to develop world models that can understand, reason and plan around physical environments.

"What you need is a predictor that, given the state of the world and an action you imagine, can predict the next state of the world," he said. "And if you have such a system, then you can plan a sequence of actions to arrive at a particular outcome."

COMP4300 – News - Nvidia GTC 2025

NVIDIA HPC SDK



DEMO

Numerical Integration using Quadrature

- **Problem Definition**
  - Computing the definite integral of sin(x) from 0 to π
  - The exact result is 2.0, which makes it easy to verify our numerical approximation
  - We use the rectangle rule for numerical integration
- **Sequential Algorithm**
  - Divides the integration interval into n subintervals
  - Evaluates the function at the midpoint of each subinterval
  - Multiplies the sum by the width of each subinterval to get the approximation
- **Parallelization Strategy**
  - Domain decomposition: the integration range is divided among processes
  - Each process handles its own portion of the integration range
  - Local results are combined using MPI_Reduce with the sum operation
  - The root process (rank 0) collects the final result
- **Performance Measurement**
  - Uses MPI_Wtime() to measure execution time for both sequential and parallel algorithms
  - Calculates and reports speedup
  - Computes the error compared to the exact analytical solution

DEMO

Numerical Integration using Quadrature

The code also demonstrates basic MPI concepts like:

- Process identification (rank and size)
- Data partitioning
- Collective operations (MPI_Reduce)
- Synchronization (MPI_Barrier)
- This implementation should give you a clear understanding of how to parallelize numerical integration problems using MPI.

DEMO

Solving a unit lower triangular system using forward substitution

- **Program Features:**
  - Generates a random unit lower triangular matrix (1's on diagonal, random values below, 0's above)
  - Creates a random right-hand side vector
  - Implements both sequential and parallel forward substitution algorithms
  - Measures and compares execution times
  - Verifies solution accuracy by calculating error
- **Forward Substitution Algorithm:**
  - In forward substitution, we solve for x in Lx = b by:
  - $x_1 = b_1$ (since $L_{11} = 1$ in a unit triangular matrix)
  - For each subsequent row i:
    - $x_i = b_i - \Sigma(L_{ij} \times x_j)$ for j = 1 to i-1
- **Parallel Implementation:**

  The MPI implementation distributes rows among processes, with each process:
  - Computing values of x for its assigned rows
  - Broadcasting each computed value immediately to all other processes
  - Using values computed by other processes for its dependencies

DEMO

Solving a unit lower triangular system using forward substitution

- **For small matrices**, the sequential version may be faster due to communication overhead
- **For large matrices**, the parallel version should show improved performance
- **The speedup will not be linear** with the number of processes due to:
  - The inherently sequential nature of forward substitution (each element depends on previous solutions)
  - Communication overhead for broadcasting solutions
  - Load imbalance (earlier processes finish earlier)
- This example illustrates the classic trade-off between computation and communication in parallel algorithms, especially for algorithms with data dependencies like forward substitution.

Why use MPI for Parallel Computing?

Scalable High-Performance Computing

- MPI enables efficient parallel computation across thousands of processors, making it the standard for solving computationally intensive problems in scientific computing, engineering, and big data analytics.
- Its ability to distribute workloads across multiple nodes allows researchers and engineers to tackle problems far beyond the capabilities of a single machine.

Flexible Communication Paradigms

- Unlike shared memory models, MPI provides robust point-to-point and collective communication mechanisms that work across distributed memory systems. This flexibility allows for:
  - Dynamic data exchange between processes
  - Efficient communication topologies
  - Precise control over data movement and synchronization
  - Compatibility with diverse hardware architectures

Why use MPI for Parallel Computing- II?

Platform and Language Independence

- MPI is a standardized and portable message-passing system that:
  - Works across different computing platforms
  - Supports multiple programming languages (C, C++, Fortran)
  - Enables code portability from desktop clusters to supercomputers
  - Provides consistent performance characteristics across different systems

Comprehensive Communication Operations

- MPI offers a rich set of communication primitives that go beyond simple data transfer:
  - Blocking and non-blocking communication
  - Collective operations (broadcast, reduce, scatter/gather)
  - Advanced datatype handling
  - Process group and communicator management
  - Complex synchronization mechanisms

Why use MPI for Parallel Computing - III?

Proven Performance in Critical Domains

- MPI is the key to high-performance computing in critical fields such as:
  - Climate and weather modeling
  - Molecular dynamics simulations
  - Computational fluid dynamics
  - Particle physics research
  - Genomics and bioinformatics
  - Financial modeling and risk analysis
- Its widespread adoption in scientific computing demonstrates its reliability, efficiency, and capability to solve complex computational challenges.

Course Topics Covered So far

- **Introduction** (parallelism concept and rationale; scales of parallelism; application areas; Moore's and Dennard's scaling laws; frequency and ILP race; power and ILP walls; multicore revolution; Top 500; current trends in supercomputing)
- **Implicit Hardware Parallelism** (peak versus effective processor performance; ILP; pipelining; superscalarity; limitations of memory system performance; memory hierarchy; caching)
- **Explicit Hardware Parallelism** (Flynn's Taxonomy; SIMD versus MIMD parallel architectures; message-passing versus shared address space; UMA versus NUMA; dynamic and static interconnection networks; evaluation metrics for static interconnection networks)
- **Message-passing parallelism with MPI** (definition; motivation and history; code compilation and execution; point-to-point communication; semantics of point-to-point communication and buffering; deadlocks and deadlock-avoiding techniques; MPI collectives, datatypes and communicators)
- **Performance Measures and Models** (parallel speedup and efficiency; parallel overheads; hardware, algorithmic, strong and weak scalability; Amdhal's law; Gustafson's law; speed-up and scaled speed-up; timer resolution and overhead)

Course Topics (II)

- **Embarrassingly parallel problems** (definition; master/slave paradigm; static versus dynamic mapping; performance trade-offs; parallel performance analysis; examples)
- **Routing and communication costs** (routing, definition and classification; communication cost model, start-up, per-hop and per-word transfer times; routing techniques: store-and-forward and cut-through; modelling the cost of collectives on static interconnection networks; example: one-all and all-all broadcasts; improving speed of communication operations)
- **Synchronous computations** (definition and examples; static partitioning of grids and matrices; recursive doubling all-gather; performance modelling; the concept of ghost layer aka halo; deadlock avoiding approaches)
- **Parallelization by Partitioning and Divide-and-conquer** (definition and examples; performance modelling; binary tree-like collective communication)
- **Parallelization by pipelining** (definition and examples; performance modelling; pipeline latency and average time per problem instance)