26 Architecture and Performance Characteristics of Modern High Performance Computers

Georg Hager and Gerhard Wellein

Regionales Rechenzentrum Erlangen der Friedrich-Alexander-Universität
Erlangen-Nürnberg, 91058 Erlangen, Germany

In the past two decades the accessible compute power for numerical simulations has increased by more than three orders of magnitude. Many-particle physics has largely benefited from this development because the complex particle-particle interactions often exceed the capabilities of analytical approaches and require sophisticated numerical simulations. The significance of these simulations, which may require large amounts of data and compute cycles, is frequently determined both by the choice of an appropriate numerical method or solver and the efficient use of modern computers. In particular, the latter point is widely underestimated and requires an understanding of the basic concepts of current (super) computer systems.

In this chapter we present a comprehensive introduction to the architectural concepts and performance characteristics of state-of-the-art high performance computers, ranging from the “poor man’s” Linux cluster to leading edge supercomputers with thousands of processors. In Sect. 26.1 we discuss basic features of modern commodity microprocessors with a slight focus on Intel and AMD products. Vector systems (NEC SX8) are briefly touched. The main emphasis is on the various approaches used for on-chip parallelism and data access, including cache design, and the resulting performance characteristics.

In Sect. 26.2 we turn to the fundamentals of parallel computing. First we explain the basics and limitations of parallelism without specialization to a concrete method or computer system. Simple performance models are established which help to understand the most severe bottlenecks that will show up with parallel programming.

In terms of concrete manifestations of parallelism we then cover the principles of distributed-memory parallel computers, of which clusters are a variant. These systems are programmed using the widely accepted message passing paradigm where processes running on the compute nodes communicate via a library that sends and receives messages between them and thus serves as an abstraction layer to the hardware interconnect. Whether the program is run on an inexpensive cluster with bare Gigabit Ethernet or on a special-purpose vector system featuring a high-performance switch like the NEC IXS does not matter as far as the parallel programming paradigm is concerned. The Message Passing Interface (MPI) has emerged as the quasi-standard for message passing libraries. We introduce the most important MPI functionality using some simple examples. As the network is often a performance-limiting aspect with MPI programming, some comments are made...
about basic performance characteristics of networks and the influence of bandwidth and latency on overall data transfer efficiency.

Price/performance considerations usually drive distributed-memory parallel systems into a particular direction of design. Compute nodes comprise multiple processors which share the same address space (shared memory). Two types of shared memory nodes are in wide use and will be discussed here: The uniform memory architecture (UMA) provides the same view/performance of physical memory for all processors and is used, e.g., in most current Intel-based systems. With the success of AMD Opteron CPUs in combination with Hypertransport technology the cache-coherent non-uniform memory architecture (ccNUMA) has gained increasing attention. The concept of having a single address space on a physically distributed memory (each processor can access local and remote memory) allows for scaling available memory bandwidth but requires special care in programming and usage.

Common to all shared-memory systems are mechanisms for establishing cache coherence, i.e. ensuring consistency of the different views to data on different processors in presence of caches. One possible implementation of a cache coherence protocol is chosen to illustrate the potential bottlenecks that coherence traffic may impose. Finally, an introduction to the current standard for shared-memory scientific programming, OpenMP, is given.

### 26.1 Microprocessors

In the “old days” of scientific supercomputing roughly between 1975 and 1995, leading-edge high performance systems were specially designed for the HPC market by companies like Cray, NEC, Thinking Machines, or Meiko. Those systems were way ahead of standard commodity computers in terms of performance and price. Microprocessors, which had been invented in the early 1970s, were only mature enough to hit the HPC market by the end of the 1980s, and it was not until the end of the 1990s that clusters of standard workstation or even PC-based hardware had become competitive at least in terms of peak performance. Today the situation has changed considerably. The HPC world is dominated by cost-effective, off-the-shelf systems with microprocessors that were not primarily designed for scientific computing. A few traditional supercomputer vendors act in a niche market. They offer systems that are designed for high application performance on the single CPU level as well as for highly parallel workloads. Consequently, the scientist is likely to encounter commodity clusters first and only advance to more specialized hardware as requirements grow. For this reason we will mostly be focused on microprocessor-based systems in this paper. Vector computers show a different programming paradigm which is in many cases close to the requirements of scientific computation, but they have become rare animals.

Microprocessors are probably the most complicated machinery that man has ever created. Understanding all inner workings of a CPU is out of the question for the scientist and also not required. It is helpful, though, to get a grasp of the high-level features in order to understand potential bottlenecks. Figure 26.1 shows a very