A Brief Introduction to the Design and Implementation of Parallel Numerical Simulation Codes

Large scale scientific computing problems associated with the numerical simulation of physical processes such as those occurring in fluid flow and structural dynamics have greatly stimulated the recent evolution of numerical algorithms suitable for parallel computing architectures. The development of efficient parallel numerical simulation algorithms is particularly difficult since computer software and hardware are also evolving at a relatively rapid pace to satisfy the ever-increasing demand for floating point arithmetic performance. Even though a diverse (almost combinatorial) range of architectural possibilities must be carefully weighed by the software architect, the papers contained in this chapter discuss a number of important recurring themes in the development of efficient numerical simulation codes:

1. **Code portability.** Owing to the overall complexity of numerical simulation codes, portability and re-usability of software has become an increasing issue for software providers. For large commercial simulation software packages, it is simply not feasible to provide a large number of code modifications to gain compatibility with each new computer architecture. Fortunately, a number of standardised libraries are now routinely used for explicit message passing (e.g. PVM and MPI) and for shared memory access (e.g. SHMEM). In addition, standardised languages such as FORTRAN-90 and C as well as object oriented languages such as C++ provide stable development environments together with parallel scientific libraries such as ScaLAPACK for parallel linear algebra tasks and many others. The performance of these libraries is extensively evaluated in subsequent papers.
2. **Minimised communication, data locality and load balancing.** When compared to local (on-board) communication, the relative cost of communication between remote (off-board) processors can be quite high. This cost of communication is strongly related to the bandwidth and latency of sending messages or cache lines between processors. Consequently, an essential ingredient in obtaining high computational efficiency is the mapping of data to memory and/or processors of the computer. This is true to a variable degree across a broad range of architectures utilising distributed memory access via message passing protocol as well as shared memory access (local or global) via cache line protocol. In addition, the work tasked to each processor should be balanced with a minimum number of synchronization points so that processors do not wait idly for other processors to catch up. Several papers contained in this chapter address this problem by explicitly partitioning the domain into a number of smaller (possibly overlapping) sub-domains using a variety of mesh partitioning techniques. These partitioning techniques divide the computational domain among processors so that the work load is approximately balanced and interprocessor communication is approximately minimised. Inside each sub-domain, the data may be reordered in memory to further improve data locality so that cache memory misses are reduced.

3. **Speedup and Scalability.** Common threads contained in the papers of this chapter are the basic motivations for using a parallel computer in large scale scientific computations. These motivations embody the expectation that by introducing more processors into the computation it will become possible to (1) solve a problem with fixed number of unknowns in a shorter amount of wall clock time (speedup), (2) solve a problem that exceeds the capacity of a single processor in terms of memory and/or computer time, and (3) to control the overall algorithmic complexity in terms of arithmetic operations as the number of solution unknowns is increased (scalability) by employing multiple meshes and processors. This latter motivation is the divide-and-conquer effect of certain domain decomposition methods and the multi-level effect of multi-grid methods. A number of techniques for obtaining motivations (1-3) are discussed in subsequent papers based on non-overlapping domain decomposition, additive Schwarz on overlapping meshes, and multi-grid. Each has relative merits depending on the relative cost of communication and computation, granularity of parallel computations, differential equation discretized, and discretization technique used. Note that for some applications that require only a few processors, parallel speedup may be the most important issue. In other applications requiring thousands of processors, scalability may become a dominate issue. Each technique must be evaluated against the desired class of computations.

The broad range of computing architectures used in actual parallel computations suggests that the standardisation of libraries for interprocessor communication, parallel linear algebra, etc. has produced an environment suitable for the development of large scale numerical simulation codes. Even so, the demands for