Foreword for the *Patterns for Parallel Software Design* Book by Jorge Ortega Arjona

The steady increases in processor speeds associated with Moore's law have improved software performance for decades without necessitating significant changes in software designs or practices. Over the past several years, however, the exponential growth in CPU speed has stalled. Increases in software performance now stem largely from exploiting parallel processing to exchange data reliably and scalably across high-speed interconnects, dynamically balance workload in computation grids, and efficiently synchronize access to shared resources. Researchers and practitioners rely on parallel processing to accelerate scientific discoveries and deliver value to users in a wide range of application domains, including high-performance scientific computing, weather forecasting, financial services, animation rendering, text mining, homeland security, and enterprise content management.

Although parallel processors and interconnects continue to improve, it remains tedious and error-prone to develop complex application and infrastructure software that can meet challenging—and changing—user requirements. This situation has yielded a "parallel software crisis," where the hardware becomes ever more capable, but the software remains hard to develop, debug, optimize, and evolve. Much of the effort expended on parallel software is spent rediscovering core concepts (such as coordination, communication, and synchronization) and reinventing common components (such as active objects, dynamic load balancers, job schedulers, message brokers, and notification engines). Moreover, despite advances in key technologies (such as concurrent programming languages, vectorizing and optimizing compilers, operating system clustering techniques, and grid computing middleware), many software developers lack experience with *how* and *when* to best apply these technologies.

Addressing the parallel software crisis therefore requires more than just adopting the latest technologies—it requires learning and applying successful parallel software *patterns* that document recurring architectures and designs and convey proven parallel software structures, algorithms, and best practices. Knowledge of patterns helps researchers and practitioners avoid rediscovering and reinventing core concepts and common components of parallel software. Patterns can also explain how and when to best apply parallel technologies.

Popular patterns (such as Adapter, Bridge, Reactor, and Strategy) have captured and guided the designs of application and infrastructure software for two decades. Many of these patterns were initially identified by developers of object-oriented graphical user interface frameworks that work in contexts where quality factors like usability, extensibility, and portability are paramount. In addition to these quality factors, developers of parallel software must also understand and apply patterns that work in contexts where low latency and high throughput, reliability, and scalability are paramount.

Over the years, isolated coverage of parallel software patterns has appeared in various conference proceedings and books. For example, the proceedings and books associated with the Pattern Languages of Programming conferences present patterns for scalable locking and threading, synchronous and asynchronous event handling, and loosely-coupled group communication. Likewise, the Pattern-Oriented Software Architecture book series presents patterns for pipeline parallelism, master/slave processing, distributed request brokering, and dynamic resource management. Until Jorge Ortega Arjona published this book on *Patterns for Parallel Software Design*, however, no single source provided such a broad and deep spectrum of architectural patterns, design patterns, and common idioms for developing parallel software.

The patterns and idioms that Jorge present in this book help resolve key parallel software challenges, such as coordinating interactions between concurrently executing tasks, partitioning parallel algorithms and data to improve performance substantially, and minimizing synchronization overhead in local and distributed shared memory. In addition to describing the structure and functionality of essential parallel software patterns and idioms, Jorge also presents many examples from a range of applications domains, including high-performance scientific computing, image processing, and animation rendering. Moreover, Jorge's detailed case studies extend the book beyond a catalog of parallel software patterns to provide keen insights into parallel software design processes and methods that help alleviate key accidental and inherent complexities in parallel software development projects.

For parallel software development to move from an art to an engineering discipline, successful practices and design expertise must be documented systematically and disseminated broadly. My colleagues and I have documented and applied patterns in a wide range of distributed and parallel application and infrastructure software, including the ACE, TAO, and Zircomp middleware. We've found that studying and applying patterns helps to

- *Facilitate reuse of architecture and design artifacts*, which reduces the effort required to develop high quality parallel software frameworks and application components. These patterns can be reused even when reuse of algorithms, implementations, interfaces, or detailed designs was not feasible due to heterogeneous software and hardware platforms.
- *Document ``best practices'' of parallel software systems*, which have traditionally resided in the minds of expert developers or buried within complex source code. Capturing the most useful strategies and tactics of parallel software in terms of patterns lowers the learning curve for new developers by giving them good role models for developing parallel software applications and infrastructure.
- *Preserve important design information*, which is often lost over time in conventional development processes, causing increased maintenance costs and software defects. Software evolution effort can thus be reduced significantly by documenting the intent, structure, and behavior of parallel software components in terms of the patterns they reify, as well as explaining how and when to best apply these components in various contexts.
- *Guide design choices for new systems* since patterns capture proven experience in a form that can be used to address new design challenges. By understanding the potential traps and pitfalls in their domains, developers can select suitable parallel software architectures, protocols, and platform features without wasting time and effort implementing solutions that are known to be inefficient or error-prone.

A thorough understanding of the parallel software patterns, processes, and methods in Jorge's book will likewise help you develop better parallel software applications and infrastructure. If you want thorough coverage of the key pattern-oriented software architectures that are shaping the next-generation of parallel software then read this book. I've learned much from it and I'm confident that you will too.

Douglas C. Schmidt
Nashville, Tennessee, USA