Parallel Software Engineering -- Goals 2000

Carl Murphy
Accord Solutions Inc
San Diego, CA 92103

Abstract

The challenge of parallel software engineering is to provide predictable and effective products and processes for a new breed of application. To do this, parallel computing must apply to all concurrent application types. Assured delivery and predictable time response are the metrics to determine success. In addition, high integrity factors (availability, reliability, security and graceful reaction to changes) are necessary. Infrastructure and life cycle support issues are important to consider when creating large scale information and command & control systems.

The approach is to define an encompassing engineering process that accounts for quantitative and predictive measures of time response, integrity and application complexity.

Engineering Process Metrics

A general principle for evaluating the potential for technology transfer is how well a technology fits into the engineering process. Advanced technology has little value added if the infrastructure is not present to use it [Murphy95A]. Too often, parallel computing finds itself at odds with the software engineering process. Engineering requires predictability and life-cycle metrics. Parallel computing has exacerbated the capability of software engineers to reliability predict time response, integrity, project and life-cycle metrics.

The difficulty of adopting parallel software technology into the engineering process is that you need a total package. Figure 1 diagrams the situation. Success means that the system delivers on all mission requirements, time response goals and integrity factors, ie, glass must be completely full [Murphy95B and Murphy95C].

Parallel software engineering must undergo significant change to respond to industry and military needs over the next five years. The essence of parallel software engineering is comprehensive and concurrent delivery of on high time, high integrity and high project value applications. However, the demands are complex ones. Parallel computers are necessary to meet them, but the foundation given by today's technology is too weak. If an adequate foundation exists, then information and command and control systems will adopt high demand applications beyond the capability of client programming.

![Figure 1. The Software Engineering Challenge](image)

The next generation of information systems will be distributed-computing collections. Multi-tiered client-server architectures are the first iteration of this generation. Although computer technology will advance along Moore's law -- a factor of four every third year -- there will still be a need for application servers for enterprise critical systems that have time response and integrity barriers. Yet, such a server increases complexity and heterogeneity, adding to the design difficulty. In spite of the functional open system
machinery available, getting interoperability and predictability of these diverse systems are one of the greatest challenges to system architects and designers. Architectural tools and components must provide effective development of the system necessary to deliver responsively to a system structure-hiding, user-centric, active performance feedback model.

Reduced information systems concepts (risc)\(^1\)

To reach the "completely full glass," architecture tools and distributed components for complex information systems must hold fast to these three "risc" architectural engineering principles.

- The first principle of risc: Client application programming cannot require detailed structural knowledge about the server organization. This gives an efficient development and life cycle, plus protection from obsolescence.
- The second risc principle: The system behind the client network connection must provide correct functioning, predictable time response and the integrity necessary to meet the corporate mission. Predictability allows top-down engineering and life cycle planning.
- The third principle of risc: The development process and product must be visible to management. Information system builders must show potential value frequently. The entire delivery infrastructure must observe the solution in increments to forecast the delivery impact. Scaling from small to full size must be predictable. Visibility allows strategic planning to deliver competitive advantages.

Effective and efficient application architectures

The goal of parallel software engineering is to meet both high time value goals and C3I mission factors within the parallel environment. Applications of any character should be feasible with performance, programmability, and portability. In order to accomplish this daunting task, parallel software engineering must concentrate on the most difficult character applications.

These are the ones that have extensive complexity, dynamic resource demands, symbolic interaction and frequent interaction, ie, the toughest problem types. However, parallel software engineering must recognize the need to integrate those results with existing legacy systems and commodity-based architectures.

The basic problem is that tools are inadequate to define and predict time response and integrity of distributed architectures. Also, needed system software components are missing. As a result, a distributed system's major cost elements are not hardware and network capital costs, they are integration, development, and life cycle infrastructure costs. Often these costs reduce the expected return on investment. Predictive tools allow architects to select and organize hardware and network capability to minimize infrastructure costs.

The tools must provide quality monitoring capability that gives management better insight into the strategic version of the system. They are to reveal development and life cycle support processes. Parallel software engineering tools should reduce risks of performance, cost, and schedule shortfalls.

Parallel Software Engineering

Additional goals for parallel software engineering research are to:

- reduce the cost and effort for development and life cycle support by finding architecturally independent design and programming models
- take a second look at commercial operating systems, languages, SQL, object techniques, visual programming, and ATM data transfer;
- demonstrate coordination languages, effective protocols, and hardware interfaces
- system supervisor and control
- architectural independence
- non intrusive entry and coordination with legacy
- interface and cooperation with industry for obtaining an effective system component set

References

\(^1\) Reduced instruction set computers (RISC) matched operational frequencies to a small set of instructions resulting in architectural speed advantage. The RISC engineering method uses a quantitative prediction of performance of a design based on microprocessor functional units and their arrangement. The risc approach should give the same quantitative prediction to distributed system design.