Resource Management

Barney McCabe (ORNL) and Hugo Falter (ParTec)

A ***scalable application*** is an application whose performance scales with the size of the computing system. To be scalable an application must make effective use of additional resources, i.e., the application must demonstrate a performance improvement that is proportional to an increase in resources. This improvement can be demonstrated by reducing the time to completion for a fixed size problem (strong scaling) or by increasing the size of the problem that can be completed in the same amount of time (weak scaling). Alternately, a scalable application can be characterized as an application whose performance is constrained by the availability of one or more resources, i.e., a scalable application is a ***resource constrained application***. Ultimately, application scalability is based to the ability of the application to manage the resources provided by the computing system.

By presenting an abstraction of a computing system, ***programming models*** emphasize the management of some resources while de-emphasizing others. Successful HPC programming models emphasize the management of the resources that are most likely to constrain the scalability of an application, while de-emphasizing the management of other resources. For example, explicit message passing models, like MPI, have been very successful in HPC because they abstract the details of inter-node communication, but emphasize the management of distributed of memory by requiring that applications encode explicit message exchanges to access remote memory.

Approaches to resource management can be categorized in two dimensions: static/dynamic and explicit/implicit. Static resource management decisions are made before execution, while dynamic decisions are made during execution. Dynamic decisions typically incur some overhead (additional use of resources) during execution but they can incorporate information about the dynamic behavior of the program. Explicit resource management decisions are written into the code for the application, while implicit decisions are implemented in the translation or runtime system. Programming models emphasize the management of some resources over others by choosing which resources require explicit management by the application developer and which can be delegated to implicit management by the underlying runtime system.

Table 1. Approaches to Resource Management

	Static	**Dynamic**
Explicit	Algorithms	Zoltan load balancing
Implicit	Register allocation by a compiler	Demand-paged virtual memory

The tradeoffs between static and dynamic approaches in resource management are relatively straightforward to evaluate. Dynamic approaches can be justified when the overhead needed to monitor resource usage and to adjust the management of these resources results in an overall improvement in application performance. These justifications are typically complicated by the fact that the costs and benefits are highly application dependent and the fact that the overhead may require a resource that is different from the resource used to measure performance improvement, e.g., the overhead uses memory and performance is measured in time to completion.

Evaluating the tradeoffs between explicit and implicit approaches is rarely straightforward. Implicit resource management decisions remove much of the burden for making resource management decisions from the programmer (moving this complexity to the runtime system) and may enhance application portability, because details regarding resources of the target platform do not need to be encoded in the application. However, because implicit approaches seek to hide the true nature of the resource, there is a chance that application developers will unknowingly use the resource in an inappropriate fashion. A simple example of this comes when programmers fail to maintain temporal locality in their data access, yielding poor virtual memory or cache performance when the existence of these mechanisms is not explicit in the programming model.

No implicit resource management strategy is ideal for all applications. There is a significant chance that any implicit resource management decision will adversely affect the scalability of an important application. In most cases, the critical resource management decisions are limited to a small portion of the application and most of the application code does not need to include explicit resource management decisions. For this reason, it is important that implementations of programming models provide programmers with the tools needed to "opt out" of the implicit management decisions as needed. As an example, compilers for procedural programming languages provide implicit management of the registers available on a CPU. Using profiling tools, application programmer can identify performance critical parts of their code and, if needed, hand code specific subroutines in assembly code, opting out of the implicit management of CPU registers provided by the compiler. Providing mechanisms to opt out of dynamic, implicit resource management decisions is typically more difficult. In the past, this has been addressed by providing hints and callbacks. Hints allow the programmer to provide explicit advice to the runtime system in advance. The runtime system uses the hints provided by the programmer to guide its management of the resources. Callbacks allow programmers to register handlers that implement explicit resource management strategies.

For the past two decades, high performance computing (HPC) has focused on increases in processing resources; although, there is general recognition that balanced increases in other resources (e.g., memory, storage, and inter-processor communication) may critically impact the ability of an application to take advantage of increases in processor resources. As we enter a time in which processor cycles are ubiquitous, the processor is unlikely to be the resource which critically constrains the performance of an application. As such, we, as a community should take this opportunity to re-consider the tools and approaches available to application developers to support them in the management of resources for scalable applications.