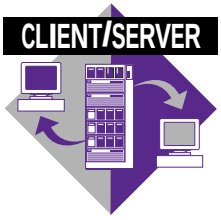
Parallel Programming Models on the IBM RS/6000 SP



By David Klepacki and Xianneng Shen

This article is the first in a series that will introduce parallel programming models on the RS/6000 SP system. It presents an overview of the various programming models and their basic features. Future articles will provide more detailed information about each programming model.

The IBM RS/6000 Scalable POWERparallel™ system is the large-scale server of the IBM RS/6000™ product line, which includes desktop workstations, group servers, departmental servers, enterprise servers, and large-scale servers. The RS/6000 SP™ system is a general-purpose parallel computer. It can operate in one of three modes:

- **Sequential/Batch mode**: More than one serial or Symmetric Multiprocessing (SMP) job can be run at the same time (Server consolidation is a good example of this mode).
- **Parallel mode**: More than one computing unit (uniprocessor or SMP) can be used at the same time for the same job.
- **Hybrid mode**: Both parallel jobs and sequential jobs can coexist on the same system.

The basic hardware components of the RS/6000 SP system are the computing units (nodes) and the interconnection network (switch). Each node is an RS/6000 server, which can be either a uniprocessor server or an SMP server. The systems are packaged into frames that come in two sizes (heights): 79 inch and 49 inch. The 79-inch frame has eight drawers and the 49-inch frame has four drawers.

In addition, the system has three types of nodes based on their physical size: uniprocessor thin nodes, uniprocessor wide nodes, and SMP high nodes. Either two thin nodes or one wide node fits into a single drawer; a high node requires two drawers.

RS/6000 SP Architecture

The RS/6000 SP is a share-nothing architecture; that is, each node has its own memory (RAM), hard drives, network connections, and operating system (AIX). Communication between nodes is carried out by sending and receiving messages over the network. This type of architecture allows high availability (engineered redundancy), flexibility (open choices of configuration), and scalability (the ability to incrementally increase performance).

In modern computer architecture taxonomy, the RS/6000 SP is also classified as a distributed-memory Multiple Instruction Multiple Data (MIMD) machine. MIMD simply means that different instruction streams of the same job that may operate

on different objects within the same dataspace can be executed simultaneously by more than one processor.

The switch network provides a low-latency and high-bandwidth communication path between the nodes. A Communication Subsystem (CSS) provides the common IP communication protocols as well as a high-speed proprietary protocol known as *user space*. Typically, technical applications employ the user-space protocol and commercial applications employ the IP communication protocols.

A complete copy of the AIX operating system runs on every node since each node is an RS/6000 server. A Parallel System Support Program (PSSP) runs on top of the AIX operating system on each node. The PSSP provides system management, high-availability infrastructure, the communication subsystem, and the virtual shared-disk system. The Parallel Environment (PE) software provides a parallel application development environment, which includes debugging and performance monitoring tools.

Parallel Programming Models

The RS/6000 SP supports three classes of parallel programming models: message-passing, data parallel, and shared memory.

Message-passing Mode

In a message-passing programming model, explicit message passing (send/receive) is used to communicate between nodes. This model provides optimal performance because it is congruent to the underlying architecture of the RS/6000 SP. In this model, an application developer makes calls to a message-passing library to invoke the various Interprocess Communication (IPC) routines within the source code.

The RS/6000 SP fully supports the Message Passing Interface (MPI), the industry-standard message-passing library. IBM strongly supports MPI standardization and has actively participated in the public MPI forum, which defines the MPI standard. The first standard definition of MPI (MPI-1) was published in May 1994.[1] Recently, the second standard definition (MPI-2) has been announced.[2] Standardization ensures portability of your applications across different parallel machines (although performance is not ensured).

The RS/6000 SP supports three classes of parallel programming models: message-passing, data parallel, and shared memory.

In addition to MPI, the RS/6000 SP supports other message-passing libraries, such as Message Passing Library (MPL), Parallel Virtual Machine (PVM), and PVMe. MPL, the first native message-passing library for the RS/6000 SP, is being phased out in support of the MPI standard.

PVM, a public domain environment developed by Oak Ridge National Lab[3], provides some features not found in either MPL or MPI-1. PVMe is an enhanced version of PVM that specifically exploits the technology of the RS/6000 SP switch hardware. Although PVM and PVMe are still widely used today, the new features specified in the MPI-2 standard will eventually cause these environments to phase out as well.

Data Parallel Programming Model

The data parallel programming model evolved from earlier parallel machine architectures known as Single Instruction Multiple Data (SIMD) machines. These machines simultaneously execute the same instruction stream on more than one processor, but operate on different objects within the same dataspace.

This programming style is very popular with the scientific and engineering communities, which are typically interested in solving mathematical problems on fixed spatial grids. Their computational technique is

[1] MPI Standard
[2] MPI-2 Standard
[3] PVM User Guide

known as *domain decomposition*, whereby the computational space is divided among the processors so that the mathematical operators can be applied to each of the disjoint subspaces in parallel. The data parallel programming model is the most natural model for this kind of computation.

The industry standard for data parallel programming in FORTRAN is High Performance FORTRAN (HPF). HPF is characterized by exclusive use of comment-based directives within the FORTRAN source code. These directives specify the distribution, layout, and alignment of the various data elements (mostly arrays) among the different processors.

The HPF compiler (or preprocessor) is responsible for the movement of data between the processors when needed. This alleviates the burden of explicitly managing the movement of data between the processors, as in the message-passing model. Therefore, the data parallel model is often referred to as an *implicit* method of programming. However, this ease-of-use with regard to the data parallel model does not come without a price on a MIMD architecture such as the RS/6000 SP.

Since the underlying architecture is MIMD and not SIMD, the data parallel programming model is not congruent to the underlying hardware; therefore, the performance will generally not be optimal. Despite this fact, the data parallel model still has advantages.

Data Parallel Model

One advantage of this model is the ability to easily change the layout of the data. In HPF, the layout of data can be easily altered with one or more `change all` commands of a text editor. For example, a one-dimensional distribution of a matrix, specified by (`BLOCK,*`) in a HPF directive, can be changed to a two-dimensional distribution among the processors by using (`BLOCK,BLOCK`) instead. The only changes needed are to the parameters inside the (relatively few) HPF directives.

On the other hand, altering the data layout in an MPI application typically would require substantial rewriting of code to accommodate the different buffer management and synchronization requirements. Therefore, the trade-off in performance by using HPF might be worthwhile for both the ease-of-use and flexibility of code changes it provides.

Besides the performance trade-off, the data parallel model suffers from other afflictions: the inability to accommodate dynamic load balancing and dynamic data structures. This should not come as a surprise, since the data parallel model is mainly interested in being applied to static domains as described above. However, newer physical models and computational techniques are forcing the need to address such issues. Although there are various research attempts to incorporate support for dynamic parallelism within HPF, none have come to fruition. In fact, the most recent HPF standard specification (HPF-2)[4] has removed some of the mandatory features dealing with dynamic redistribution, since none of the HPF compiler vendors could provide such functionality efficiently.

The industry standard for data parallel programming in FORTRAN is High Performance FORTRAN (HPF).

HPF Compilers

Today, several HPF compilers are available for the RS/6000 SP. IBM produces the xlHPF compiler; Applied Parallel Research, Inc. produces the xHPF preprocessor; and the Portland Group produces the pgHPF preprocessor. These third-party vendors supply preprocessors to retain the use of IBM's xlf compiler and all of the optimizations that come with using it.

For the C language, there is an ANSI® standard specification known as Data Parallel C (DPC) as well as a widely used (but not a standard) language known as High Performance C (HPC). HPC is public

[4] HPF-2 Standard Document

domain and attempts to mimic the features and functionality of HPF by using similar comment-based directives. All of these issues for HPF also apply to HPC.

Like HPF, DPC has its origins from the days of the SIMD architectures. It is a direct offshoot of the C language pioneered by the now defunct Thinking Machines Corporation. In fact, many applications written in C can run with minor alterations on the RS/6000 SP using a DPC compiler. Although IBM does not produce a DPC compiler, it is available for the RS/6000 SP via a third-party software vendor, Pacific-Sierra Research, Inc.

The ANSI standard DPC language is not based upon the exclusive use of comment-based directives. Rather, *shape distribution* is introduced, in which data distribution is specified in the source code by using left subscript variables. Although the syntax of this implementation is quite different, the issues involved in the data parallel model remain unchanged.

Shared Memory Model

The third type of programming model is the shared memory model, or more accurately the Virtual Shared Memory (VSM) model, since the present discussion is in the context of a distributed-memory parallel architecture such as the RS/6000 SP.

The VSM model is the ultimate extrapolation of parallel programming to a single virtual address space. Although the underlying hardware contains separate and distinct memory address spaces due to the multiplicity of individual RS/6000 server nodes, the VSM programmer can ignore this fact and assume a single (logical) address space. In fact, this programming model is basically the same model that is used when developing uniprocessor workstation applications; the only difference is the comment-based directives that identify independent parallel operations.

This model differs from a directive approach of a data parallel model such as HPF in one important way: the VSM model is not at all concerned with data distribution among the processors. The only required directives are those needed to identify independent operations within the source code, whether they are independent iterations of a loop or independent sections of code. The VSM programmer does not worry about how to partition and distribute the data among the processors as in HPF, nor about managing buffers and synchronization as with MPI.

VSM is usually the fastest and easiest model to produce a running parallel code. However, as before, since the underlying hardware is not shared memory, VSM will not yield optimal performing applications. The trade-offs involved are very similar to the trade-offs associated with the data parallel model.

> ***Virtual Shared Memory (VSM) is usually the fastest and easiest model to produce a running parallel code.***

The VSM environment must simulate a shared memory environment on top of the physically distributed memory environment. Again, this does not come without a price. This simulation requires time and resources in order to achieve the illusion of shared memory for the programmer. When a processor makes reference to a data element that is not local to its address space, the VSM system must perform the necessary operations to find and retrieve the valid data within the machine and transfer it to that processor. Furthermore, these operations are happening continuously throughout the lifetime of the job and on many processors simultaneously.

As you can see, these issues can become complex. In a future article, we will explore these issues in greater detail.

Note: A VSM environment is available today for the RS/6000 SP from a third-party vendor, Myrias Computing Technologies. Their VSM software environment is called the Parallel Application Management System (PAMS), which has been running on the RS/6000 SP since April 1997. Although the performance is not competitive with an equivalent message-passing application, the

performance is quite acceptable given the simplicity of its use. Furthermore, initial testing up to 64 nodes has indicated that PAMS provides a scalable shared memory environment for many types of applications. Lastly, PAMS runs on any node configuration, whether the nodes are thin, wide, or high, and automatically adjusts the load to compensate for the performance differences of the underlying hardware.

The next article in this series will take a closer look at the issues involved with MPI programming on the RS/6000 SP. It will also include a discussion of available development tools (both IBM and third party) and a look at the new features in MPI-2.

David Klepacki, *IBM Corporation, 522 South Road, Poughkeepsie, NY 12601. Dr. Klepacki has been working in IBM's POWERparallel Systems Group since its beginning in 1991 as a computational physicist and scientific applications specialist with emphasis on performance benchmarking. Today, in addition to his technical endeavors, he also manages the parallel software tools segment for the technical marketing branch of the RS/6000 Division. Dr. Klepacki's current interests include performance programming, scalable parallel algorithms, scalable I/O, and portable high-performance computing tools. He holds a PhD in Theoretical Nuclear Physics from Purdue University as well as an MS in Electrical Engineering from Syracuse University.*

Xianneng Shen, *IBM Corporation, 522 South Road, Poughkeepsie, NY 12601. E-mail: xshen@us.ibm.com. Dr. Shen is a senior marketing support representative in the RS/6000 Executive Briefing Center. He has a BS and an MS in Electrical Engineering from the University of Electronic Science and Technology of China, an MS in Computer Engineering from Syracuse University, and PhD in Electrical Engineering from Syracuse University*