Sciddle 4.0, or, Remote Procedure Calls in PVM

Peter Arbenz[1], Walter Gander[1], Hans Peter Lüthi[2], and Urs von Matt[2]

[1] Institute of Scientific Computing, Swiss Federal Institute of Technology (ETH), 8092 Zürich, Switzerland, email: `[arbenz,gander]@inf.ethz.ch`
[2] Swiss Center for Scientific Computing, Swiss Federal Institute of Technology (ETH), 8092 Zürich, Switzerland, email: `[luethi,vonmatt]@scsc.ethz.ch`

Abstract. SCIDDLE 4.0 makes it possible to use remote procedure calls (RPCs) within the PVM environment. Parallelism is achieved through asynchronous RPCs. No explicit message passing is necessary. Rather all data transfers occur within the RPCs.
In SCIDDLE, an application is decomposed into a client process and an arbitrary number of server processes. Servers are special PVM tasks that are ready to execute RPCs from their clients. Servers can also start other servers themselves. Thus the topology of a SCIDDLE application can be described by a tree structure.
SCIDDLE applications enjoy the safety and ease of use of RPCs. They are also extremely portable as PVM becomes available on more and more platforms. In this paper we show that the overhead introduced by SCIDDLE is minimal and can be neglected for applications with large data sets.

1 Introduction

The communication environment SCIDDLE [1, 2, 8] has proved to be a valuable tool for high-performance network computing. Most notably, it has been used for computations on network connected Cray supercomputers in quantum chemistry research [5].

A SCIDDLE application adheres to the client-server paradigm. A client process can attach several server processes which will provide services through RPCs. A client may have multiple servers, but servers cannot communicate with each other. However each server can start additional server processes. Therefore the overall communication pattern of a SCIDDLE application is given by a tree structure.

A SCIDDLE application does not need to send and receive messages explicitly. Rather all interprocess communication takes place through RPCs, and the user moves within paradigms well-known from sequential programming. In many cases, the server program is almost identical to the corresponding part of the sequential version of the program.

SCIDDLE provides an interface definition language to define the semantics of the RPCs. A compiler translates this interface definition into communication stubs for the client and server processes. Error checking is available both at compile-time and at runtime.

SCIDDLE 3.0, which was used in [11], relies on UNIX socket connections to implement asynchronous RPCs. In recent years, however, PVM has become a de-facto standard as an environment for parallel applications based on the message passing programming model [4, 10]. Its wide acceptance has lead numerous computer vendors to provide high-performance PVM implementations. For the Intel Paragon and the IBM SP/2, PVM implementations exist that sit directly on top of the native message passing interface [3]. Intra-node PVM messages are transfered via the shared memory on the Cray J90 SuperCluster [7]. The main reason for providing SCIDDLE 4.0 was the need to be able to exploit these optimized PVM implementations.

From the application programmers point of view SCIDDLE does not change. Features like spawning and termination of tasks are still possible, as they are supported by PVM. The PVM user, on the other hand, will be able to exploit the RPC concept when programming distributed applications. From a software engineering point of view, this adds a major benefit to the PVM system.

2 Software Overview

SCIDDLE uses a simple declarative language to specify the RPCs exported by a server. Fig. 1 presents an example of an interface definition. Constants may

```
INTERFACE Example;
  CONST N = 100;

  TYPE A = ARRAY [N] OF REAL;
       R = RECORD
             i: INTEGER;
             a: A;
           END;

  PROCEDURE P (IN s, n: INTEGER; INOUT a [1:s:n]: A;
               OUT r: R): ASYNC;
END
```

Fig. 1. Remote Interface Definition

be defined and used as symbolic array dimensions. User-defined types may be constructed from arrays and records. Procedures come in two flavors, synchronous and asynchronous. Each procedure parameter is tagged with a direction attribute. Parameters are always copied in the respective directions.

As SCIDDLE is designed in particular for parallel distributed numerical applications, it provides special array handling support for easy distribution of subarrays to multiple servers. A subarray can be selected by attributing array parameters with views. A view is, like an array section in MATLAB [6], a triple of the form [begin-index:stride:end-index]. The view components are either constants or the names of other integer parameters passed in the same call.

The SCIDDLE stub compiler translates a remote interface definition into the appropriate PVM communication primitives. In the example of Fig. 1 the procedure P is declared asynchronous. Thus the client stub provides the two subroutines

```
int invoke_P (int s, int n, A a, int sid);
int claim_P  (A a, R *r, int cid);
```

A call of invoke_P initiates an asynchronous RPC on the server sid. The input parameters are sent to the server, and a call identifier cid is returned as the function result. invoke_P returns as soon as the parameters are safely on their way to the server.

As soon as the client is ready to receive the results from an asynchronous RPC it calls the subroutine claim_P. The call identifier obtained from invoke_P is consumed, and the output parameters are retrieved.

If the procedure P had been declared synchronous the server stub would only provide the subroutine

```
int call_P (int s, int n, A a, R *r, int sid);
```

A call of call_P blocks the client until the server has completed its calculations. Such a blocking RPC may be useful for quick calculations, but no parallelism can be obtained in this way.

The server process must implement the procedure

```
void P (int s, int n, A a, R *r);
```

The server stub receives the input parameters from the client, calls P, and sends the results back to the client.

The SCIDDLE runtime system also offers additional subroutines for starting and terminating server processes. Multiple ongoing asynchronous RPCs can be managed conveniently by means of call groups [8].

3 Experiments

The performance of SCIDDLE 4.0 is analyzed by means of a so-called “ping-pong” experiment. We measure the time it takes for a message of n bytes to travel from a process A to a process B and back. Obviously this test can be implemented as an RPC in SCIDDLE, where both the size of the input and the output argument is n bytes.

Our experiments compare the performance of SCIDDLE 4.0 versus PVM. In the first test the client and server processes run on two Cray J90 nodes connected by a HIPPI network. Figure 2 shows the latency (transfer time from user space to user space) as a function of the message size. The fragment size in PVM was chosen to be 32768 bytes.

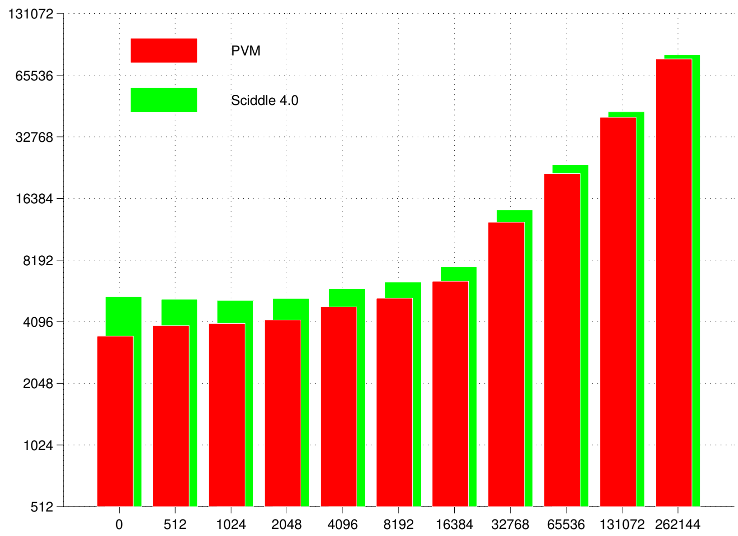

Fig. 2. Latency (μsecs) versus Message Size (Bytes) between two Cray J90 Nodes

Since SCIDDLE 4.0 is built on top of PVM we observe an increased communication startup time. On the other hand the asymptotic bandwidth (about 3.3 MBytes/sec) does not change.

In the second test the client and server processes run on two different processors on the same Cray J90 node. Again, Fig. 3 shows how the transfer time increases with the message size. Compared to Fig. 2 the latency is roughly cut in half, and the asymptotic bandwidth approaches 7.5 MBytes/sec.

In both tests we do not only measure the raw communication (network) time but also the time required to pack and unpack a message. On the Cray J90 this overhead is relatively small but on other computers the delay can be significant [9].

4 Conclusions

SCIDDLE 4.0 offers a simplified and easy-to-use interface to PVM. In SCIDDLE parallelism is provided through the use of asynchronous remote procedure calls. The dependencies between clients and servers can be described by a tree structure. This model is more restrictive than in PVM where arbitrary communication patterns are allowed. However protocol errors are much easier to detect in SCIDDLE where checks are performed both at compile-time and at runtime.

PVM has been optimized by Cray Research and other vendors for their computer systems, and SCIDDLE 4.0 provides easy access to these optimizations. Even though another layer is added on top of PVM, we only observe a minor

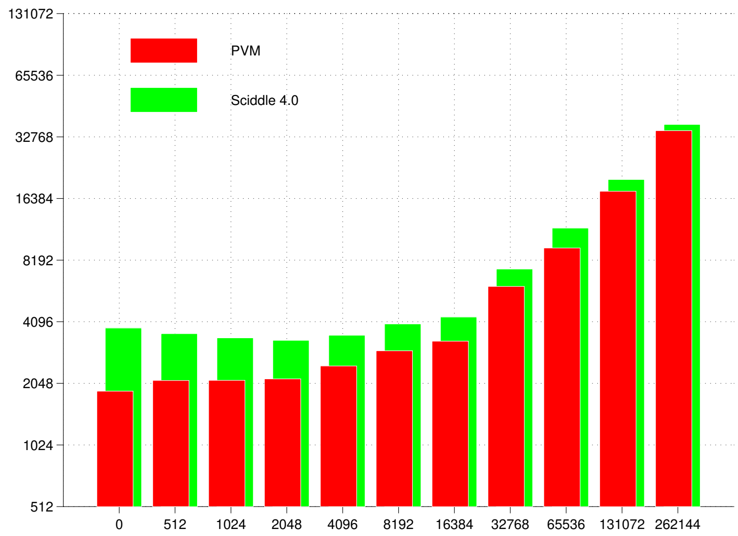

Fig. 3. Latency (μsecs) versus Message Size (Bytes) on a single Cray J90 Node

degradation of performance, which is by far outweighed by the safety and ease of use of SCIDDLE.

References

1. P. Arbenz, H. P. Lüthi, J. E. Mertz, and W. Scott, Applied Distributed Supercomputing in Homogeneous Networks, International Journal of High Speed Computing, 4, 87–108, 1992.
2. P. Arbenz, H. P. Lüthi, Ch. Sprenger, and S. Vogel, SCIDDLE: A Tool for Large Scale Distributed Computing, Concurrency: Practice and Experience, 7, 121–146, 1995.
3. H. Casanova, J. Dongarra, and W. Jiang, The Performance of PVM on Massively Parallel Processing Systems, Tech. Report CS-95-301, University of Tennessee, Computer Science Department, Knoxville, TN, August 1995, `http://www.cs.utk.edu/~library/TechReports/1995/ut-cs-95-301.ps.Z`.
4. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, PVM, Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing, The MIT Press, Cambridge, MA, 1994.
5. H. P. Lüthi and J. Almlöf, Network Supercomputing: A Distributed-Concurrent Direct SCF Scheme, Theoretica Chimica Acta, 84, 443–455, 1993.
6. The MathWorks Inc., MATLAB, High-Performance Numeric Computation and Visualization Software, Natick, Massachusetts, 1992.
7. H. Poxon and L. Costello, Network PVM Performance, Cray Research Inc., Software Division, Eagan, MN, unpublished manuscript, June 1995.

8. Ch. Sprenger, User's Guide to SCIDDLE Version 3.0, Technical Report 208, ETH Zürich, Computer Science Department, December 1993, `ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/2xx/208.ps`.
9. V. Strumpen, The Network Machine, Diss. ETH No. 11227, Swiss Federal Institute of Technology (ETH), Zürich, 1995, `ftp://ftp.inf.ethz.ch/pub/publications/dissertations/th11227.ps`.
10. V. Sunderam, G. A. Geist, J. Dongarra, and R. Manchek, The PVM Concurrent Computing System: Evolution, Experiences, and Trends, Parallel Computing, 20, 531–545, 1994.
11. St. Vogel, J. Hutter, T. H. Fischer, and H. P. Lüthi, Direct SCF Structure Optimization of Large Molecules on Networks of Workstations, International Journal of Quantum Chemistry, 45, 665–678, 1993.