Release Notes for Version 2.3 of SR

Gregg M. Townsend

Department of Computer Science
The University of Arizona

October 7, 1994

Version 2.3 of SR includes the following changes:

- A reimplementation of arrays designed for better performance
- Support for multiprocessing on an Intel Paragon
- A new **`SRgetopt`** library procedure and an interface to XTANGO
- Portability improvements, including rewritten IBM RS/6000 code
- Other minor enhancements

These changes are described later in more detail. Also, a small number of bugs have been fixed; these are itemized in the **`Versions`** file.

The base version of SR, version 2.0, is documented in *The SR Programming Language: Concurrency in Practice*, by Gregory R. Andrews and Ronald A. Olsson (Benjamin/Cummings, 1993, ISBN 0-8053-0088-0). Information about concurrent programming in general can be found in *Concurrent Programming: Principles and Practice*, by Gregory R. Andrews (Benjamin/Cummings, 1991, ISBN 0-8053-0086-4). These books can be ordered through your local bookstore or by telephoning 1-800-950-2665 or 1-415-594-4400. Manual pages for the compiler and other components are provided with the SR system.

Array Reimplementation

To improve the performance of scientific applications, the implementation of arrays in SR has been redesigned. The changes affect both the layout of arrays in memory and the code generated to access array elements.

A one-dimensional array consists of a header followed by data. Before, multidimensional arrays were just arrays of arrays, each array being self-contained with its own header. Now, a multidimensional array consists of a single header followed by all the data elements; the data elements are packed contiguously, with no embedded headers. This reduces memory consumption and corresponds (except for the single header) to the layout used by C.

The new memory layout also allows SR to generate code that can be better optimized by a typical C compiler.

The SR Programming Language comes from the Department of Computer Science, The University of Arizona, Tucson, Arizona 85721 USA. The implementation is available by anonymous FTP from ftp.cs.arizona.edu, and the SR Project may be reached by sending electronic mail to sr-project@cs.arizona.edu.

Incompatibilities

Some new limitations are imposed when dealing with *subarrays*. A subarray is an incompletely indexed array: For example, if **`A`** is a three-dimensional array, then **`A[i,j,k]`** is an element of the array but **`A[i]`** or **`A[i,j]`** are subarrays. With the change in array representation, the memory area addressed by a subarray no longer has the form of a valid array. Previous restrictions on the use of slices now also apply to subarrays, which can be considered a form of slice.

Few programs are expected to be affected by the tighter restrictions. Subarrays can still be used as values or as targets of assignment; these are accomplished by copying. Subarrays can no longer be passed as **`ref`** parameters, but they can still be used as **`val`**, **`var`**, or **`res`** parameters.

Due to the change in array representation, external functions that deal with multidimensional arrays will need to be rewritten. As before, the address of an array's data is passed to the external function, but multidimensional arrays no longer contain subarray headers within the data. The layout of singly-dimensioned data is unaffected.

Other performance issues

A new compiler option, **−T**, suppresses the code at the top of every loop that checks the global ''timeslicing'' counter and periodically effects a context switch. With **−T** set, one process can maintain exclusive control of the CPU indefinitely, starving other processes that could otherwise run. The **−T** option can improve the performance of programs with tight loops.

When speed is paramount, the performance of an SR program can be maximized by following as many of these guidelines as possible:

- build SR with **`CFLAGS = -O -DNDEBUG`**
- do not enable MultiSR when building SR
- compile SR programs with **`-O -T`**
- use constant-sized arrays
- pay close attention to the algorithm and coding

Beware, however, of problems caused by defective optimizers; SR is normally built and tested without optimization.

Scanning Changes

The input function **`scanf`** now accepts formatting specifications of the form **`%[...]`**. This form matches input from the specified character set in the same manner as the corresponding C format. Although this form is documented in the SR book, it had not been implemented.

A field width of 1, which is required for scanning into a **`char`** variable, is now the default when scanning into a char variable.

Library Changes

Two contributed packages have been added to the SR library, which previously contained only **SRWin**. *Man* pages are available for these packages.

- **SRgetopt** parses command arguments in a manner similar to C's **getopt**.
- **SRanimator** provides an interface to the XTANGO animation package, which is available from Georgia Tech. A paper describing **SRanimator** is provided in PostScript form in the file **ps/sranimator.ps**. Several sample programs are provided in **examples/sranimator**.

Intel Paragon Support

Version 2.3 of SR adds support for the Intel Paragon architecture. A Paragon is a collection of independent Intel i860 processors, each with private memory, connected in a mesh by a fast communications network. In SR, each processing node is a different ‘‘physical machine’’, and a distributed programming model is used.

SR programs run in the default partition. This is specified by the environment variable NX_DFLT_PART or by the –**pn** command line argument as described in the Paragon User's Guide.

Resources are created on virtual machines; virtual machines are created on physical machines (processor nodes) by **create vm() on** *X*. The expression *X* is normally an integer in the range 0 through *N-1*, where *N* is the number of available nodes. Alternatively, *X* can be a string form of such an integer or a string previously mapped by a call to **locate()**.

The number of available nodes and the node number on which a VM is running can be found using the Paragon runtime functions **numnodes()** and **mynode()**. These are made accessible by declaring them external within SR:

```
external numnodes() returns int
external mynode() returns int
```

The following example prints out **Hello from node** *X* on every node in the default partition.

```
resource hello()
    external mynode() returns int
    write("Hello from node ", mynode())
end

resource main()
    import hello
    external numnodes() returns int
    const N := numnodes()
    var vm_cap [0:N-1] : cap vm
    var hello_cap [0:N-1] : cap hello
    fa i := 0 to N-1 ->
        vm_cap[i] := create vm() on i
        hello_cap[i] := create hello() on vm_cap[i]
    af
end
```

It is possible to create more than one virtual machine on a processor node. This is inefficient, however, because idle VMs consume processor time while busy-waiting for incoming messages. Also, a 16 MByte node only has enough memory to create three to five VMs.

Other Portability Improvements

Several minor modifications accommodate system changes on some platforms or increase the portability of the SR implementation. The most notable modifications are:

- The context switch code for the IBM RS/6000 has been rewritten and is much more reliable than in the past.
- Code is included for running SR under NetBSD.
- Minor changes have been made to fix problems associated with the Linux operating system.
- **`SRWin`**, the X-Windows interface, has been modified slightly to allow it to run on the Dec Alpha architecture.

Earlier Language Extensions

This section describes the other changes that have been made to the SR language since the first printing of the SR book. Most of these were introduced in version 2.2.

Input from Capabilities

In SR 2.0, an operation capability could be used as a destination for an invocation but not as a source. This asymmetry has been mostly removed. The remaining limitations ensure that an input statement requires at most a single exchange of messages with a different virtual machine.

An operation capability can now replace an operation name in an input statement, without regard to how the capability was created, subject to meeting *either* of the following two restrictions:

- the input statement has just one arm and no **`by`** or **`st`** (but it may have an **`else`**), or
- the capability references an operation in the current resource.

Capabilities are also allowed in **`P`** and **`receive`** statements; these are shorthand forms of input statements meeting the first restriction.

As an example, a bag of tasks can be encapsulated in a resource by exporting an operation name that represents the bag. Processes in other resources can both contribute and withdraw tasks by using a capability for the exported operation.

The question mark operator (**`?`**) can be used to query the number of pending invocations available through a capability.

A **`noop`** capability is treated as an operation that is never invoked. Hence, input from a **`noop`** capability never succeeds. Querying a **`noop`** capability with the **`?`** operator always returns 0. Attempting to query or input from a **`null`** capability—or a capability for an operation that no longer exists—produces a run-time error.

A capability used with **`P`**, **`receive`**, or **`?`** can be any arbitrary expression. For syntactic reasons, a capability used in an **`in`** statement is limited to the form of an identifier, possibly qualified and/or subscripted.

Dynamic Operations

New operations can be created dynamically. Any of the following expressions creates one instance of a new, anonymous operation and returns a capability for that operation:

`new (`*optype_id***`)`**
`new (op` *operation_specification***`)`**
`new (op` *operation_specification operation_restriction***`)`**
`new (sem)`

The operation can be invoked or serviced until the resource or operation is destroyed.

The operation referenced by capability *ocap* can be destroyed by executing

destroy *ocap*

Only dynamically created operations can be destroyed. Note that in this case **destroy** and not **free** is the opposite of **new**; this is consistent with the destruction of resources using capabilities. A subsequent invocation, query, input, or destroy of a destroyed operation produces a runtime error.

Arrays of operations can be created using any of the forms

new ([*subscripts***]** *optype_id***)**
new ([*subscripts***] op** *operation_specification***)**
new ([*subscripts***] op** *operation_specification operation_restriction***)**
new ([*subscripts***] sem)**

The array of operations referenced by *ocap_array* can be destroyed by executing

destroy *ocap_array*

Elements of an array can also be destroyed individually; destruction of any individual element renders wholesale destruction of the array illegal, because that would try to destroy the individual element a second time.

Process Handoff

The call **nap(0)** yields control to another process of equal or higher priority, if there is one. This change was made in version 2.1; prior to that, **nap(0)** had no effect.

Compatibility

Version 2.3 introduces minor changes at the source code level. Few programs are likely to be affected. The changes are as follows:

- As described earlier, subarrays are no longer lvalues and are subject to the same restrictions as slices.
- In the **SRWin** package, the **winCursor** and **winPixel** datatypes were changed to allow **SRWin** to run on the Dec Alpha architecture.

SR does not attempt to provide binary compatibility across versions; all programs must be recompiled. Resources compiled by earlier versions cannot be linked with version 2.3 resources or libraries.

The installation process is similar to that of version 2.2. A **library** directory has replaced the former **srwin** source directory, and the method of configuring for X Windows has changed. The method for accessing **SRWin** from SR programs has not changed.

Acknowledgements

This release of SR reflects the efforts of many people. Oliver Spatscheck ported SR to the Intel Paragon and provided documentation. Pedro de las Heras Quiros ported SR to NetBSD. Vince Freeh redesigned the array code and participated in its implementation. Edgar Greuter supplied code for **scanf**. Ron Olsson contributed **SRgetopt**. Stephen Hartley contributed the XTANGO interface and most of the associated examples. Alex Zhao provided the changes to **SRWin**.

Appendix: Implementation Limitations

This section lists some limitations of the current implementation. Some of these could reasonably be considered bugs. (This section has not changed significantly since version 2.1.)

The Programming Environment

- Source files are generally expected to reside in the current directory; explicit paths elsewhere don't always work.
- Keywords in strings or comments occasionally make *srgrind* think it saw a **`pb`** (procedure begin), typesetting a new name in the margin.
- The *SRWin* library does not function well on a Sequent Symmetry.

Problems Seen at Compilation Time

- Slices cannot be used with the swap (**`:=:`**) operator.
- Builtin procedures having string result parameters (**`read`**, **`get`**, **`scanf`**, **`getarg`**, **`sprintf`**) cannot accept slice expressions for those parameters.
- Unions are really implemented as records; no storage is shared.
- Some older C compilers disallow the expression **`f().m`** where **`m`** is a member of the struct returned by function **`f`**. If C lacks this support, the corresponding expression in SR will fail to compile.
- SR uses a multipass compiler. If it finds an error on one pass, it skips the later passes, with the consequence that not all errors are reported at once. Errors in **`spec`** sections may be reported once for each import.

Problems Seen at Execution Time

- The bounds of an array passed by **`ref`** are those of the underlying array, not those declared for the parameter.
- Null and noop capabilities cannot be created for resources that export arrays of operations if any of those arrays has more than three dimensions.
- Synchronization expressions are not recomputed when their values are changed by the actions of other processes (e.g. by altering global variables or sending to operations referenced by a **`?`** operator).
- Destroying a resource is not an atomic operation. If processes in two resources each simultaneously try to destroy the other resource, one or both of the processes can be killed, leaving one or both resources incompletely destroyed.
- In most situations, the process with the highest priority is executed. However, processes that are blocked waiting for messages become unblocked without regard to priority.

Problems Involving Distributed Programs

- If a distributed SR program is found using the shell's search path, but is not in the current directory, then *srx* generates an incorrect network file path and virtual machines cannot be created.
- Because of network pipeline delays, output from different virtual machines may be misordered, even if it is explicitly synchronized.
- Error messages can sometimes occur, due to race conditions, during normal termination of distributed programs. For example, these can be caused if X exits while Y is sending to it.
- Virtual machines don't always shut down cleanly if an SR program is run in the background under the Bourne shell. This is noticed when running *srv* under *sh* or *at.*

Undetected Programming Errors

Not all errors are detected by the SR compiler or runtime system. These errors can manifest themselves in various ways including segmentation faults, other runtime errors, or even misleading diagnostics. Some of the more common such errors are these:

- Arithmetic overflow is not detected by either the compiler or runtime system.
- The use of an uninitialized variable is not detected, nor is the use of a **`ptr`** or **`ref`** parameter on a different virtual machine.
- Values of enumeration types are not checked at execution time.
- Unreachable code is not diagnosed.
- No warning is given if an invocation is still pending when an operation ceases to exist.