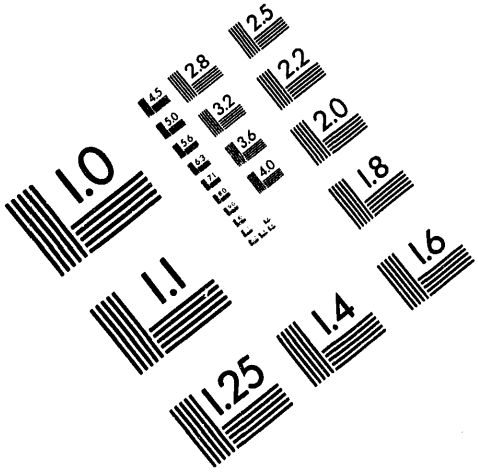
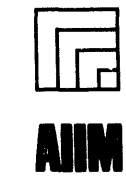
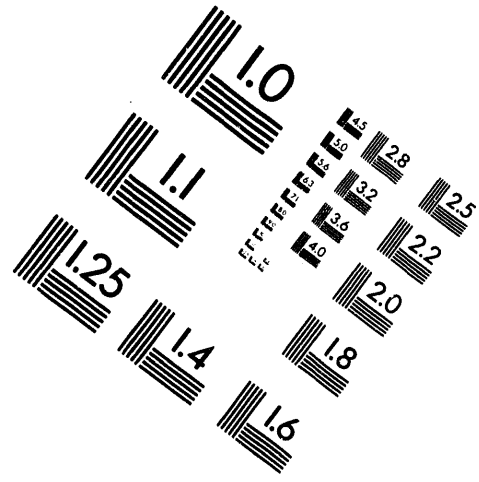
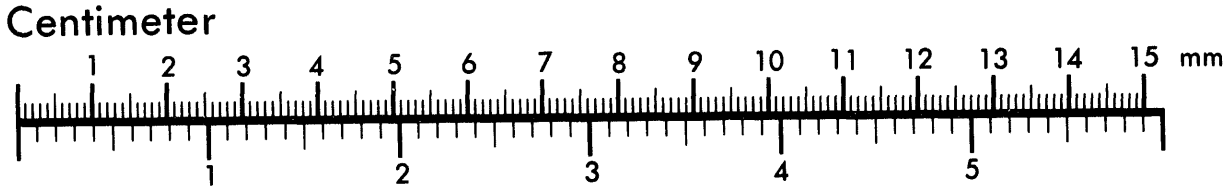
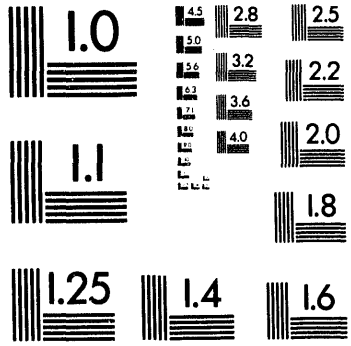


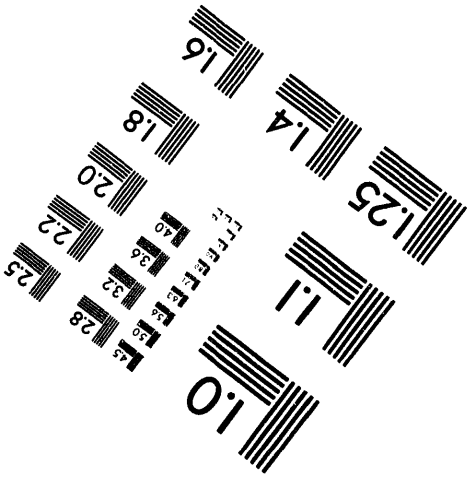


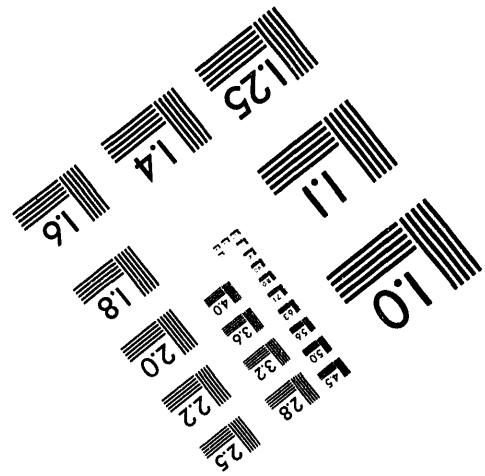
Association for Information and Image Management
1100 Wayne Avenue, Suite 1100
Silver Spring, Maryland 20910
301/587-8202











MANUFACTURED TO AIIM STANDARDS
BY APPLIED IMAGE, INC.

1 of 1

SAND94-1228C
Conf-941118--2

Split-C and Active Messages under SUNMOS on the Intel Paragon

Rolf Riesen* Arthur B. Maccabe†

April 1994

Abstract

The compute power of the individual nodes of massively parallel systems increases steadily, while network latencies and bandwidth have not improved as quickly. Many researches believe that it is necessary to use explicit message passing in order to get the best possible performance out of these systems. High level parallel languages are shunned out of fear they might compromise performance.

In this paper we have a look at one such language called Split-C. It fits into a middle ground between efforts such as High Performance Fortran (HPF) and explicit message passing. HPF tries to hide the underlying architecture from the programmer and let the compiler and the run time system make decision about parallelization, location of data, and the mechanisms used to transfer the data from one node to another. On the other hand, explicit message passing leaves all the decision to the programmer. Split-C allows access to a global address space, but leaves the programmer in control of the location of data, and offers a clear cost model for data access.

Split-C is based on Active Messages. We have implemented both under the SUNMOS operating system on the Intel Paragon. We will discuss performance issues of Split-C and make direct comparisons to the Thinking Machines CM-5 implementation. We will also scrutinize Active Messages, discuss their properties and drawbacks, and show that other mechanisms can be used to support Split-C.

This work was supported by the United States Department of Energy under Contract DE-AC04-94AL85000.

*Organization 1424; Sandia National Laboratories; Albuquerque, NM 87185-1109; email: rolf@cs.sandia.gov

†Department of Computer Science; The University of New Mexico; Albuquerque, NM 87131 email: maccabe@cs.unm.edu

1 Introduction

There are many proposals and suggestions on how to "best" program massively parallel machines. Depending on the goals of software developers and end-users, "best" can take on several and, unfortunately, contradictory meanings. Most tradeoffs are between performance and ease of use. Massively parallel systems are the current state of the art and are priced accordingly. It would therefore be wasteful not to utilize these systems to the highest degree possible.

Split-C, in the spirit of the C programming language, gives full access to the hardware and the available resources, while providing an extra level of abstraction. With Split-C, a programmer doesn't have to worry about explicit message passing anymore. At the same time the structure of the distributed memory is not hidden, and lets developers optimize algorithms and control the location of each data item.

In this paper we explore the price of this convenience. We have implemented active messages (AM), the foundation of Split-C, under the SUNMOS operating system for the Intel Paragon. This gives us the opportunity to measure the performance of AM and Split-C programs and compare them to results published earlier for the Thinking Machines CM-5 [vECGS92] [CDG+93].

Specifically, we are interested in the effect of the low latency and low bandwidth network of the CM-5 on user level applications, versus the comparatively high latency and high bandwidth network of the Intel Paragon.

The paper is structured as follows. In the next section we give some background information about Split-C, AMs, the CM-5, the Intel Paragon, and SUNMOS. In section 3 we explain the steps and decisions we made to bring Split-C and AMs to the Intel Paragon. Section 4 directly compares the CM-5 implementation with ours. We give exact numbers for sending and receiving AMs as well as regular messages, and measure an application program running on the CM-5 and the Intel Paragon. In section 5 we discuss the pros and cons of AMs, based on our experience with them on the Intel Paragon. Section 6, finally, closes the paper with some conclusions and ideas for further work.

2 Background

Split-C [CDG+93] is a superset of the C programming language. The main feature is access to a global address space distributed among the processors of a massively parallel machine. The language differentiates between global and regular pointers. Dereferencing a global pointer is more costly than dereferencing a regular pointer, since it will result in a remote data request, if the data is residing on another node. In contrast to languages such as High Performance Fortran (HPF), it is always clear whether data is local or remote. Therefore, a programmer can make decisions based on a clear cost model for memory references.

Split-C gets its name from the possibility of split-phase access to remote data. That is, through the use of special assignment operators a remote data access can be initiated and then completed in the background, while the program continues processing. At the point where the data is actually needed, a **sync** operation is performed to ensure the data has arrived. If the data is not available yet, **sync** will block until the data arrives. The following code fragment illustrates this:

```
double *global gPtr;
double local, x;

/* Split phase assignment */
local:= *gPtr;

/*
** other processing... state of
```

```
** "local" unknown.
*/

sync();

/*
** The global value pointed to by
** "gPtr" is now available in the
** variable "local".
*/

x = local * 3.0;
```

The advantage over explicit message passing is, that the node holding the data does not have to explicitly synchronize with the requesting node. There is no need to anticipate and receive the request. This simplifies programming parallel systems greatly.

To accomplish this, Split-C uses active messages (AM) [vECGS92]. An active message consists of a function address and up to four 32 bit parameters. The node receiving the AM will execute the function specified and pass it the parameters. The functions executed, called AM handlers, are intended to be short and are used to send requested data back, update counters, or store data on the remote node. AMs are quite flexible and can be used to implement other message passing schemes or can serve as the foundation for parallel languages.

The fundamental property that makes AMs possible, is that the application and the environment are homogeneous. All nodes have the same physical characteristics and the programs running on them are the same. This means that the start address of a specific function is the same on all nodes the application is currently running on. The same is true for statically allocated data objects. This makes it possible to determine the address of a variable on one node and read that address on another node to get the value of the same variable on the remote node.

The Split-C compiler and library are available via ftp from the University of California at Berkeley. The compiler is based on gcc and has been ported to a variety of architectures.

The original implementation of Split-C ran on a Thinking Machines CM-5. The CM-5 uses Sparc microprocessors as its node computing elements. Optionally, four vector units per node can be installed to enhance floating-point performance. The topology connecting the nodes is a fat tree. Access to the network can be done from the user level; no traps into the kernel are required. This makes very low latencies below $10\mu s$ possible. The bandwidth of the data network is between 5 MB/s and 20 MB/s, depending on the "distance" between the communicating nodes. This is low compared to the Intel Paragon [PJ92] [BEK93].

Timing studies for this report were performed on an Intel Paragon XP/S. The individual nodes are connected in a two-dimensional mesh. The achievable bandwidth of the network in our configuration is 175 MB/s peak. Each node consists of two i860-XP microprocessors and 16 MB or 32 MB of memory.

The vendor supplied operating system for the Paragon is OSF1/AD using the Mach microkernel. Our implementation of Split-C and AMs runs under the Sandia and University of New Mexico (SUNMOS) operating system. SUNMOS can replace OSF on the compute nodes and offers much higher network bandwidth and lower latency than OSF. SUNMOS consumes less than 2% of the available physical memory and is therefore well suited to run applications requiring large ammounts of physical memory [Int92] [BEK93].

3 Implementation

We have modified the SUNMOS kernel to send and receive active messages. Sending an AM is very similar to sending a zero length message, since there is no data packet. The format and length of an AM is always the same, so we were able to pack the function address, sixteen bytes of parameters, as well as the parameter types and the source node rank into the message header. Figure 1 depicts an AM header.

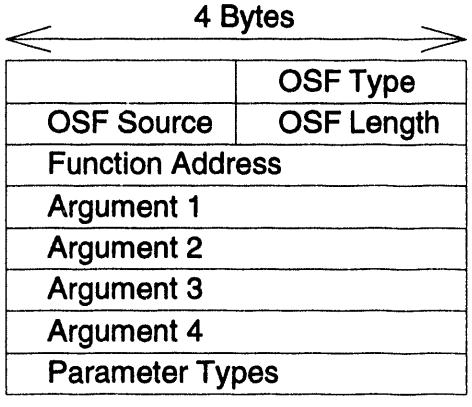


Figure 1: Active Message Header

The original AM implementation on the CM-5 exploits the fact that the Sparc CPU uses the same type of register to pass integers and floating-point numbers to functions. The calling conventions for the i860 CPU require that the type of the arguments is known on the receiving node. We encode the type in the last word of the message header. Opposed to the CM-5, accessing the communication hardware requires a trap into the kernel.

Any message arrival causes an interrupt on the receiving node. For AMs the kernel sets up a new context to run the AM handler. Instead of returning to the user level where the interrupt occurred, execution continues in the AM handler. When it is done, execution resumes where the interrupt occurred without another trap into the kernel (the original context can be restored at the user level). If an AM handler is already running, the kernel queues new arrivals until the current handler finishes. Therefore, AM handlers execute atomically with respect to each other. Figure 2 shows the timing diagram for active messages under SUNMOS on the Intel Paragon.

Most of the time in porting Split-C to SUNMOS and the Intel Paragon was spent retargeting the compiler for the i860 and rewriting the Split-C library. The original library is very much CM-5 oriented and has to deal with the idiosyncrasies of that architecture. For example, messages are limited to a length of six words (24 bytes) including the destination address. Also, care has to be taken choosing the correct network to avoid the possibility of deadlock. Unfortunately, many of these aspects are visible through the AM layer and make it rather tricky to port AMs to another architecture.

Since we were striving for the best possible performance, we removed all CM-5 dependencies in the library, as long as it did not require a change to the Split-C compiler. AMs were only used where necessary and we took advantage of specific SUNMOS features wherever performance or convenience demanded it. For example, a `bulk_get` call results in the posting of a non-blocking receive and an AM to the node that currently owns the data. The AM handler on the remote node then sends the data using the SUNMOS send primitive. When the data arrives on the receiving node, the kernel increments a flag as part of the receive operation. Thus, there is no second message required to notify the receiving node that the data has arrived.

The mechanisms in the kernel are very much like the ones used to implement signals and `hrecv`. As a matter of fact most of the code is shared and conditionals make sure execution continues at the correct place.

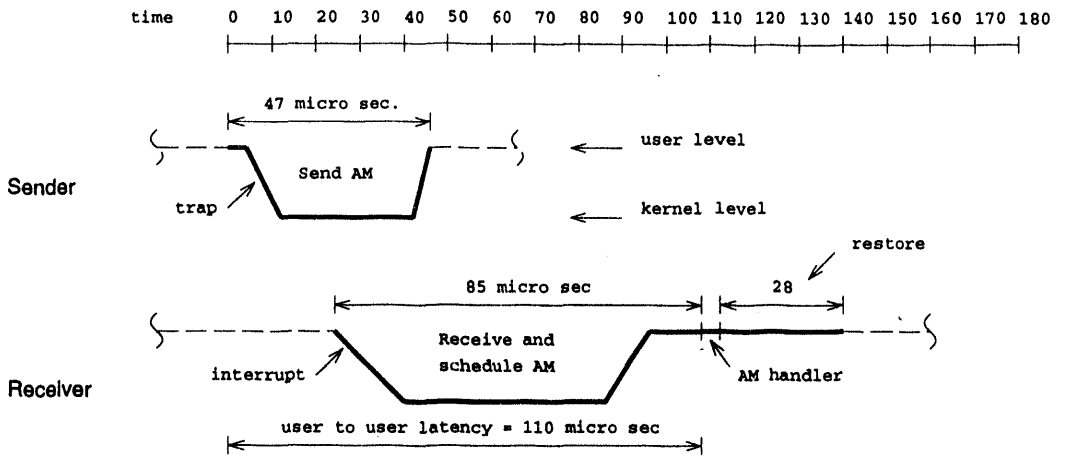


Figure 2: Active Message Timing

The Intel NX call **hrecv**, is a preposted receive. When a matching message arrives, a user specified signal handler is activated. In a sense they are very similar to AM, except that the receiving node determines what function gets executed. The **hrecv** call could be used to implement AM. A program starts by preposting a **hrecv** with a pointer to an AM dispatcher. When an AM arrives, the dispatcher gets activated which then calls the function specified by the AM. When the function returns the dispatcher preposts another **hrecv**, so the arrival of the next AM reactivates the dispatcher again.

By taking advantage of SUNMOS features in our version of the Split-C library we were able to support Split-C with only one of the forty or so functions of the CM-5 AM layer. Furthermore, we have plans to eliminate AMs altogether and port Split-C to Puma[WMR+94] [MW93]. Puma, the successor of SUNMOS, offers openings into a user program's address space, called portals. AMs are used by Split-C mainly to store and retrieve data from remote nodes. Portals offer that functionality and are faster than the current implementation of AM under SUNMOS. The reason is, that the Puma kernel will handle the request without first passing control to the user level.

4 Performance

Figure 2 shows the timing diagram for sending and dispatching an AM under SUNMOS. In [vECGS92] the authors report a time of $1.6\mu s$ to send an AM on the CM-5. That is about thirty times faster than the SUNMOS number of $47\mu s$. The need to trap into the kernel to access the communication hardware is responsible for the overhead.

The situation on the receiving end is, at first sight, even worse. Eicken, Culler, et al. report $1.7\mu s$ to dispatch an incoming AM. That is about a tenth of the time it takes to save context in SUNMOS when an interrupt occurs. However, the main reason this can be done so quickly on the CM-5 is that the AM layer polls for incoming messages. For benchmarking purposes it is easy to setup a scenario where an AM gets received and dispatched in record time, due to the fact that the receiver was ready and waiting for the incoming message.

In order to see how AMs really perform we have to turn to applications that use them. Matrix multiply is a favored benchmark in the scientific computing community. It is

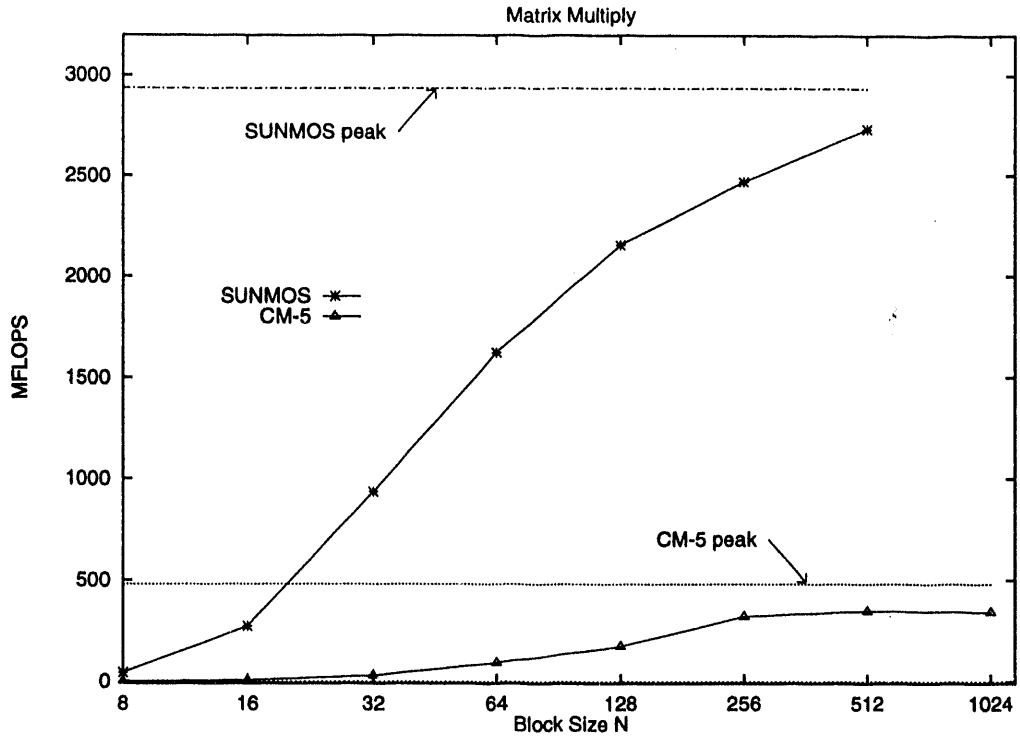


Figure 3: Matrix multiply on 64 nodes. Each matrix is 8x8 blocks.

the basis or part of many algorithms and its performance is a good indicator for a certain class of applications. The Split-C distribution from UCB comes with the example program `mm`. The matrices are subdivided into blocks and distributed among the available processors. A hand coded assembly routine is responsible for multiplying individual blocks on each node. The Split-C function `bulk_get` is used to transfer blocks between nodes. Results for the CM-5 have been reported in [CDG+93] and are reproduced in Figure 3.

The test was run on 64 nodes without vector processing units. Each of the three matrices A, B, and C consist of 8×8 blocks. The x axis of the graph shows increasing block sizes from 8×8 to 1024×1024 double precision floating-point numbers. The y axis measures the total number of Million Floating Point Operations per Second (MFLOPS) the program achieved.

In order to run this program under SUNMOS, the assembly code had to be replaced. We chose to use the BLAS library function `dgemm`, available for the Paragon, as a replacement. The function expects the matrices in Fortran style format (column major order). Therefore, we have to transpose the blocks before and after the call to `dgemm`. The timings in Figure 3 for the SUNMOS version of `mm` include the transpose times. The `mm` program requires the storage of two blocks on each node in addition to the storage requirements for part of the three matrices A, B, and C. Since each node has only 32 MB of memory, we were not able to fit the required portions of the 8×8 blocks with 1024×1024 elements onto our nodes. Note that it would be possible to fit 16×16 blocks of size 512×512. However, this would skew the direct comparison, and we have it left out for that reason.

Communication overhead in `mm` is low, and

the astounding performance of the Paragon is mainly due to the floating-point performance of the i860-XP. The more than ten times faster network bandwidth of the Paragon helped to speed up the transfer of the larger blocks.

On the CM-5, AMs are used at the lowest level and other message passing mechanisms are built on top of it. We believe that this is not a generally acceptable practice. To confirm our suspicion, we measured the transfer time of a 16 byte message. That is the amount of data that is transferred by a single AM. Figure 4 shows the result.

The timings for an AM and a 16 byte transfer are very similar. It costs less to send an AM since no checking of the parameters is needed. The four arguments are passed by value and the function address is presumed to be valid. The Memory Management Unit (MMU) of the receiving node will prevent execution of code that is not in the user's address space. The regular send needs to make sure that the sending process has read access to the 16 bytes. However, the savings are dwarfed by the time it takes to trap into the kernel and to send the data.

To simply receive a 16 byte message is even faster than dispatching an AM, since processing the AM takes time and an additional context save and restore is needed for the AM handler. One could argue that the preposting of the receive has to be taken into account also, which would swing the pendulum back in favor of AMs again.

The hardware of the Intel Paragon transfers larger packets at a higher bandwidth than smaller ones. It is also easier to amortize the high startup cost of sending a message, when several data items are grouped and transferred together. The program `mm` does a good job at that. Using AM as a lowest level layer on the CM-5 is only beneficial because the packet length is limited to 24 bytes. It is not applicable for architectures such as the Paragon and SUNMOS, where the maximum packet length is not limited.

There is a fear in the high performance community that any attempt to avoid explicit message passing will degrade performance and make the raw capabilities of the underlying hardware inaccessible. The `mm` program serves as a good example that this need not be the case. The maximum reported compute rate for the hand coded assembly routine is between 6.5 and 7.5 MFLOPS. This means the highest attainable rate on 64 CM-5 nodes is 480 MFLOPS. The maximum performance of `dgemm` for a 512×512 double precision matrix is 45.9 MFLOPS [Int93]. That results in 2937 MFLOPS on 64 nodes. Both these peak numbers are shown in Figure 3. Both, the CM-5 and the Paragon implementation, come very close to their respective maxima. Therefore, a message passing solution using the same core matrix multiply cannot be significantly faster.

While it is possible to transfer small items of a few bytes length in Split-C, a good programmer will group them together as much as possible and transfer them with a single request. The same is true for explicit message passing. However, Split-C can make the task easier since no explicit synchronization between nodes is needed.

5 Critique of Active Messages

AM are a very powerful and general mechanism that is going to influence further research into message passing paradigms and solutions to conquer the latency problem of MP machines. In this section we discuss some of the drawbacks of AMs and show ways how a language like Split-C, that depends on AMs, can be implemented without them.

The most cumbersome aspect of AMs in their current form is their closeness to the

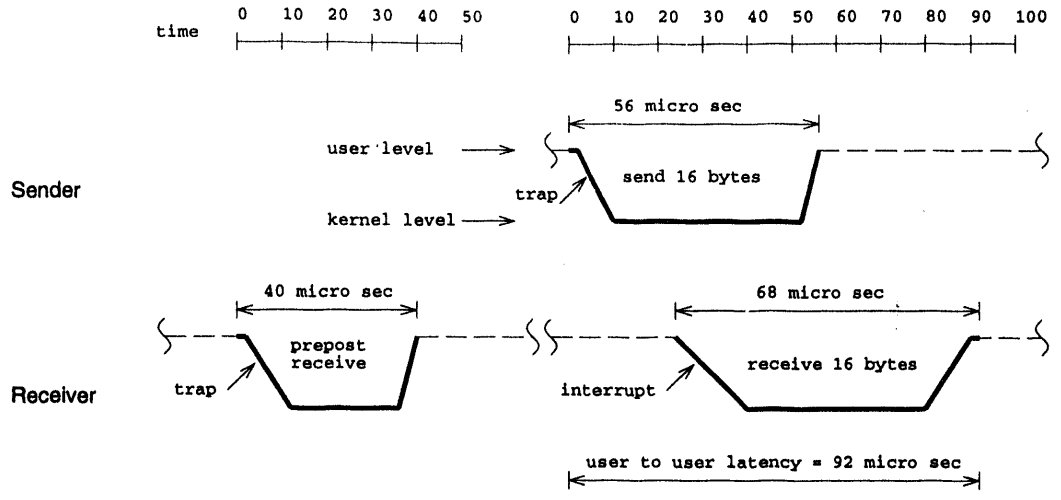


Figure 4: 16 Byte Message Timing

CM-5. This is an implementation issue that needs to be addressed before AMs can become more widespread. It should be possible for AMs to deliver more than the four words of parameters to a function that can be done on a CM-5. In most cases this is not a severe limitation, but there is no reason to inhibit performance on machines that do not have the 24 byte packet limitation of the CM-5.

In their most general form, AMs take a variable number of arguments of various types, up to a total of 16 bytes. Any type information is lost at the sending node. Architectures that pass parameters through integer and floating-point registers need that information on the receiving end. Therefore, a uniform way has to be found to pass the type of the arguments to the function that sends the AM.

There are other restrictions of the CM-5 architecture that carry through the AM layer. For example, on the CM-5 an AM handler can use the `CMAM_reply` function and activate another AM handler on the sending node. To prevent deadlock, the second AM handler is not allowed to send yet another AM to the the receiving node. This issue is irrelevant in the SUNMOS implementation, since sending an AM is not a blocking operation (the SUNMOS kernel never blocks).

There are calls to poll and wait for messages that are not needed in interrupt driven implementations of AMs. All in all, for AMs to become available and uniform on other architectures, the interface needs to be cleaned up and freed of as many machine dependencies as possible.

The are more fundamental problems, though. AMs only work because of the homogeneousness of the environment and address space of an application. Some machines, such as the Intel Paragon, make it possible to run applications on a mixture of nodes that have different memory sizes. A program running on a 16 MB node cannot easily access information on another node in the same application that has 32 MB of memory installed.

It is also very difficult to debug program that use AMs, since the arrival and the amount and type of work done by the AM handler is non-deterministic. With explicit message passing the point at which the state of a node changes, is under programmer control. The receiving node also has control over what portions of memory can be altered by

a message arrival. This is not so for AMs. Here, the sending node determines when and how state changes. Coupled with network latencies, this makes detecting a bug on the receiving node very hard.

A single threaded program using AM inherits some of the complexity associated with multithreading. For example, the updating of a local counter has to be done atomically now, if an AM handler could be activated that accesses the same counter. Split-C allows the atomic execution of functions. Since AM handlers run atomically with respect to each other it is easy to sequentialize access to a counter by sending an AM to do that, even if the counter resides on the same node. Again, this works well on the CM-5 where latencies and dispatch times are negligible. In our implementation there are better ways to control access than incurring a trap into the kernel.

AM handlers are meant to be short functions that do not block. However, what an AM handler does is under the control of the sending node. Therefore, it is possible for a node to spend many CPU cycles on behalf of other nodes without any way to control the amount or type of work done.

The Split-C team at Berkeley has ported Split-C to the Paragon under the OSF operating system. Preliminary tests indicate that `mm's` performance under that implementation is about half that of the SUNMOS implementation. While SUNMOS delivers better communication than OSF, compute performance is nearly identical. Since `mm` is not communication bound, where is the difference coming from? The hardware, the compiler, and the underlying BLAS function `dgemm` are all the same. The difference is, apart from the OS, in the Split-C library. The Split-C team attempted to use `hrecv` to implement their library. Since that function does not work properly under the current releases of OSF, they had to resort to a polling strategy. To exploit `dgemm` fully, very large matrices should be multiplied. That means that the CPU spends most of its time inside the BLAS routine without polling for new AMs. This, of course, delays data delivery to the node that sent the AM.

This is a good example to show that an interrupt driven implementation is more realistic than one that polls for new AMs. While the latter has the appeal of very short receive and dispatch times, it is not a good approach when the receiving node is supposed to do other work than simply polling the network.

We mentioned earlier that AM are a powerful and general mechanism. However, it is not clear yet how that power can be utilized and harnessed. Split-C uses AM only to access data on remote nodes. There are other mechanisms available for that purpose that avoid the above mentioned problems.

6 Further Work and Conclusions

The availability of two i860's on each node gives us the chance to experiment with various setups. We already have the ability to send and receive regular messages with one CPU while the second CPU continues processing. It should be possible to handle AMs in a similar fashion. This would leave the main CPU uninterrupted while AMs are being handled.

We have already seen that Split-C can be implemented without AMs. Work on Puma suggests that readmem and writemem portals are enough to support Split-C. This should further decrease latency, since the Puma kernel will handle the data requests. There will be no time wasted returning to the user level [WMR+94].

More work with more complex applications than `mm` is needed to further define the tradeoffs between Spilt-C and explicit message passing.

In conclusion, we find that Split-C offers a nice level of abstraction over explicit message passing without incurring any undue performance penalties. The much higher performance of the Intel Paragon over the CM-5 for the matrix multiply example is mostly due to the better floating-point processing capabilities of the i860 over the Sparc CPU.

For the `mm` application used in this report, AM latency has not been a major factor. However, it is important to note that polling for AM is not a good solution, since that can cause other processors to block unnecessarily. The overhead for processing an interrupt can easily be amortized by another node processing data instead of waiting for the next polling cycle on the remote node.

7 Acknowledgments

We wish to thank T. Mack Stallcup and Michael C. Proicou of Intel's Super Computing Division who helped us tame the big Paragon. We would also like to thank the Split-C group at the University of California at Berkeley for providing us with CM-5 numbers and answering our questions.

References

[BEK93] Tilman Bönniger, Rüdiger Esser, and Dietrich Krekel. CM-5, KSR1, Paragon XP/S: a comprative description of massively parallel computers on the basis of a catalog of classifying characteristics. Technical Report KFA-ZAM-IP-9320, Forschungszentrum Jülich Gmbh, Zentralinstitut für Angewandte Mathematik, D-52425 Jülich, Germany, October 1993.

[CDG+93] David E. Culler, Andrea Dusseau, Seth Copen Goldstein, Arvind Krishnamurthy, Steven Lumetta, Thorsten von Eicken, and Katherine Yelick. Parallel programming in Split-C. In *Proceedings of Supercomputing '93*, pages 262–273, November 1993.

[Int92] Intel Corporation. *Paragon OSF/1 User's Guide*, September 1992.

[Int93] Intel Corporation. *Paragon Basic Math Library Performance Report*, October 1993.

[MW93] Arthur B. Maccabe and Stephen R. Wheat. The PUMA architecture: An overview. Technical report SAND93-1372, Sandia National Laboratories, 1993.

[PJ92] John Palmer and Guy L. Steele Jr. Connection machine model CM-5 overview. *IEEE*, pages 474–483, 1992.

[vECGS92] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser. Active messages: A mechanism for integrated communication and computation. In *Proceedings of the 19th International Symposium on Computer Architecture*, Gold Coast, Australia, May 1992. ACM Press.

[WMR+94] Stephen R. Wheat, Arthur B. Maccabe, Rolf Riesen, David W. van Dresser, and T. Mack Stallcup. PUMA: An operating system for massively parallel systems. In *Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences*, pages

56–65. IEEE Computer Society
Press, 1994.

DISCLAIMER

DATE
FILMED
10/5/94
END