

Messengers and Navigational Programming

**Summary of Research
University of California, Irvine**

Processes capable of autonomous migration have been studied extensively in the context of mobile agents. One of the main objectives of the *Messengers* project is to demonstrate the advantages of migrating processes (as compared to stationary processes using message-passing) in the context of high-performance scientific computations on networks of standard workstations or PCs [17, 43]. *Messengers* [2, 31] is a runtime environment that allows processes, written in C, to migrate at arbitrary points from one machine to another using explicit hop statements. After reaching the destination machine, each process continues executing with the instruction following the hop. The style of programming that results when applications are written using self-migrating processes is referred to as *Navigational Programming*. The main contributions of the Messengers/Navigational Programming project, which is a collaboration between the University of California, Irvine, and the Jet Propulsion Laboratory, Pasadena, CA, include the following:

1. Applications written using Navigational Programming have a structure that mirrors the flow of a corresponding sequential program; as a result, they are easier to develop, understand, and maintain.
2. Navigational Programming supports programs that can take advantage of the collective memories of multiple machines without requiring any parallel programming; this is referred to as *Distributed Sequential Computing*
3. Navigational Programming allows the parallelization of certain programs that are generally considered unparallelizable; it employs a concept of *Mobile Pipelines* for this purpose

The following paragraphs explain the above claims in more detail:

1. Applications written using Navigational Programming have a structure that mirrors the flow of a corresponding sequential program; as a result, they are easier to develop, understand, and maintain.

To illustrate this idea, consider the following simple program:

```
v1 = diag(A)
v2 = f1(B,v1)
v3 = f2(A,v2)
```

Now assume that A and B are large matrices, each residing on a different machine, n_1 and n_2 . To make this program work in such a distributed environment using traditional message-passing, it must be rewritten as follows:

```

if (rank = n1)
  v1 = diag(A)
  send(v1, n2)
  recv(v2, n2) ←
  v3 = f2(A,v2)
else if (rank = n2)
  → recv(v1, n1)
  v2 = f1(B,v1)
  send(v2, n1) —
end if

```

Note that the program became not only considerable longer but its flow has been completely altered. To understand the original flow, one must read the code in the order indicated by the arrows. The corresponding program using Navigational Programming is the following:

```

v1 = diag(A)
hop(n2)
v2 = f1(B,v1)
hop(n1)
v3 = f2(A,v2)

```

Note that this program preserves the original structure and control flow; the only modification is the insertion of the necessary navigational statements to migrate to the machine holding the array to be accessed. This “algorithmic integrity” of Navigational programs vis-à-vis the corresponding sequential programs is what makes them easier to develop and maintain than message-passing programs [41].

2. Navigational Programming supports programs that can take advantage of the collective memories of multiple machines without requiring any parallel programming; this is referred to as Distributed Sequential Computing

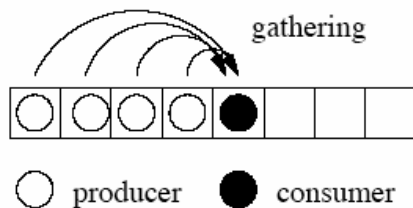
If the amount of data required by a programs exceeds the available memory on a machine, paging overhead takes over and ultimately destroys performance. A programming technique called Distributed Sequential Computation (DSC) was introduced in [29, 35] to address this problem. The idea is to distribute the data over the collective memory of the network, and to have the program reference data by hopping to the appropriate node. Note that the program is still sequential, only the data it uses is distributed. As long as the total size of the collective memory exceeds the working set of the program, paging overhead is eliminated, which results in dramatic performance improvements [42]. This solution is scalable, in the sense that the program runs efficiently as long as the network size matches the amount of memory required by the program. The technique is also quite simple to apply: the only change required of the original program is inserting hop statements at the data partition boundaries.

3. Navigational Programming allows the parallelization of certain programs that are generally considered unparallelizable; it employs a concept of Mobile Pipelines for this purpose

To support this strong claim, consider the following simple program:

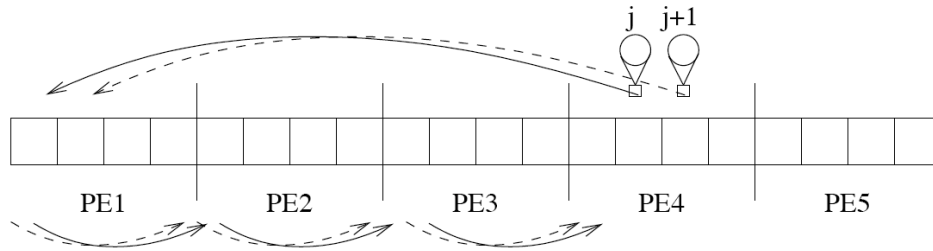
```
do j = 2 to n
  do i = 1 to j-1
    a[j] = (a[j]+a[i])*j/(j+i)
  end do
  a[j] = a[j]/j
end do
```

Note that to compute any element $a[j]$ requires all preceding elements of that array, i.e.:



Such left-looking algorithms are generally considered unparallelizable using conventional message-passing techniques. With Navigational Programming, the program can easily be parallelized as follows. First, the array is distributed over k nodes. The code is then augmented with navigational statements to visit the nodes holding the relevant portions of the array. In particular, to compute an element $a[j]$, the program copies that element and hops to node 1, where it performs the computation $a[j] = (a[j]+a[i])*j/(j+i)$ using all elements $a[i]$ located on that node. It then hops to node 2 and continues the computation with the elements $a[i]$ located on that node. This is repeated until the migrating process arrives back at the node where it started, i.e., the node holding $a[j]$, and it stores the now computed element in its location of the array.

The crucial insight is that the computation of any $a[j]$ does not happen on the node holding that element and thus does not require all the preceding elements to be moved there for the purpose of the computation. Instead, the computation is distributed over all nodes holding the required portions of the array. Consequently, the process responsible for computing any given $a[j]$ does not need to wait until all the processes computing the preceding elements have finished; instead, it can pick up $a[j]$ and start computing by hopping to node 1 as soon as the process responsible for computing $a[j-1]$ has hopped away. Thus all these processes follow each other in what has been termed a Mobile Pipeline [46]. The following figure illustrates this principle for two processes:



This keeps all the nodes busy concurrently and thus results in a parallel implementation of the above algorithm. The modifications to the sequential program are minor; they include the insertion of hop statements and the spawning of each iteration of the outer loop as a separate migrating process.

Other Advantages

In addition to the above main highlights, the Messengers/Navigational Programming has demonstrated other important advantages of strong migration of processes. One of these is the ability to parallelize existing sequential programs *incrementally* because the structure and organization of the parallel navigational code is very similar to the original sequential control flow [45, 47].

Another advantage is its inherent suitability for *Paradigm-Oriented Distributed Computing*, where the system provides the framework for a number of commonly used computational paradigms, such as a bag-of-tasks, branch-and-bound, or individual-based simulation. The user only needs to specify application-specific sequential code, while the underlying infrastructure handles the parallelization and distribution. The Messengers system provides the necessary enabling technology for this approach because it is inherently capable of adapting to a fluctuating number of machines, without requiring the user to manage the resources explicitly [25, 44].

A third promising area for migrating computations are *sensor networks*. Because of the limited computational, communication, and memory capabilities of sensor nodes, applications must be very flexible and adaptable to such specialized environments. Building on the work of others, such as the Agilla system (Washington U.), Messengers can provide a high-level environment to support Navigational Programming over sensor networks. These issues are the subject of current and planned future research of the Messengers group.

References

All papers references in this document may be found on the Messengers home page:
<http://www.ics.uci.edu/~bic/messengers/index.html>