Alcom-FT Technical Report Series
ALCOMFT-TR-03-42

# Computational models for Web- and Grid-based computation *

J. Gabarró[1] and A. Stewart[2] and M. Clint[2] and E. Boyle[2] and I.Vallejo[1]
[1] Dep. LSI, Universitat Politècnica de Catalunya,
Jordi Girona, 1-3, Barcelona 08034, Spain.
and
[2] School of Computer Science, The Queen's University of Belfast
Belfast BT7 1NN, Northern Ireland.

September 9, 2003

**Abstract**

Two models of web- and grid-based computing are proposed. Both models are underpinned by the notion of computing with "approximate" data values. The models utilise an asynchronous form of communication which blocks neither sender nor receiver. The idea of computing with approximate data is illustrated using a number of examples including a client server web computation and a partial eigensystem co-algorithm. The results of some preliminary experiments showing the feasibility of the approach for computing partial eigensolutions are presented.
**Keywords**. CPOs, approximations, Web and Grid computation, UNITY, co-algorithms.

## 1 Introduction

The areas of web and grid computation have been driven largely by technological advances – however, such computations have not been well supported by appropriate abstractions. In practice this means that design is often realised at a low-level. Notable exceptions are mobility theories (see, for example, the $\pi$-calculus or ambient calculus [1]) which provide a means of reasoning about dynamically generated links. In this paper a different aspect of Web- and Grid-based computing, namely, computing with partial information, is explored. The notion of processing partial information is intrinsic to web computing – for example, it is normal to display incomplete web pages. In a similar way, (multiple) iterative methods can be used to generate a series of partial approximations to a required fixed point. The convergence rates of individual methods may be improved by exchanging, at appropriate points in the computations, the set of available approximate solutions.

Traditional forms of communication overwrite a memory location – the receiver has no knowledge of the nature of the incoming data. However, in specialised situations, communications may *improve* (rather than just overwrite) the value stored in a location. Grab and Go communication [7, 8] is based on the notion of utilising the best available approximation

*Work partially supported by: Spanish CICYT under grant TIC2002-04498-C05-03 (Tracer), IST program of the EU under contract IST-2001-33116 (Flags), Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT) and the European Social Fund.

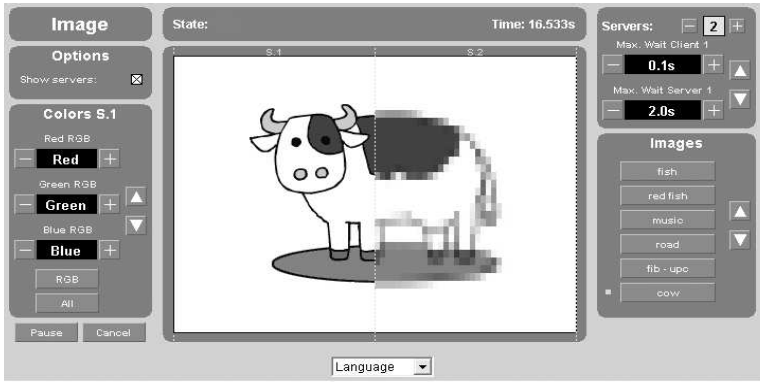


Figure 1: A Java simulation of a web-based computation. Two servers send approximations of parts of a single image. The first server has transmitted completely the first half while the other server has transmitted only a crude approximation of the second half.

immediately (rather than waiting for pending communications). Thus synchronization overheads can be avoided. In this approach approximations are viewed as elements of CPOs. A CPO [5] comprises a domain $D$ of values, a least element, $\perp$, an ordering relation $\sqsubseteq$ and a least upper bound operator $\sqcup$.

Grab and Go communication is used to underpin two models for web- and grid-based computing. The models are described in UNITY [3] and can be viewed as a variants of the inflationary fixed point problem (common meeting time) [3].

A model of web computing and an integration example are given in §2 and §3. A co-algorithm approach to exploiting grid resources is described in §4. An example of a co-algorithm which computes partial eigensolutions is given in §5. Some preliminary results are reported in §6.

## 2 A model of web-based computation

Consider the situation in which a client computer displays and processes information stored on, or generated by, servers. A server may forward information to the client piecemeal. At a given instant the client may only have received partial information from the servers (see Fig 1). In the context of web computing it is desirable that this partial information should be displayed and perhaps processed by the client.

A simple web scenario can be modelled by the generation, by two servers - *server*1 and *server*2, of two streams of input information and a computation, by a *client*, which processes both of these data streams. Let the servers *communicate* information sets $f$ and $g$, respectively. The sets $f$ and $g$ are constructed piecewise from two static sequences, $ssf$ and $ssg$. In practice $ssf$ and $ssg$ may be representations of either (i) persistent data, decomposed into packets for transmission or (ii) data directly generated by computation – the latter possibility suggests the use of sequences to represent $ssf$ and $ssg$. For example, a large image may be broken down into segments for transmission, or a live video image may become available in discrete ordered frames. This situation is dynamic and, at a particular instant, only *parts* of the complete server information may be available. Data available for transmission is inserted in either $f$ or $g$. This nondeterministic situation can be described in UNITY by:

$$\begin{array}{lllll} & \{send\ f\} & ssf, f := tail(ssf), & f \cup \{head(ssf)\} & \text{if } ssf \neq null \\ [] & \{send\ g\} & ssg, g := tail(ssg), & g \cup \{head(ssg)\} & \text{if } ssg \neq null \end{array}$$

At some instant the client will input this information ($f \cup g$) through a port variable $p$ and, perhaps, apply some function $h$ to it. The best data available to the client can be generated by combining the new data $f \cup g$ with previously received data stored in $p$:

$$\begin{array}{lll} & \{\textit{client receive}\} & p, f, g := \bigsqcup(p, f \cup g), \{\}, \{\} \\ [] & \{\textit{client apply}\} & z := h(p) \end{array}$$

For example, a better approximation to an image may be generated by adding new image fragments to a previously available partial image. In this situation a read operation by the client has the effect of *augmenting* the data stored in a variable rather than overwriting it. A client port variable is used to represent the best data approximation available. This variable is represented by an element of a CPO: no information is modelled by a least element, $\perp$, while the addition of new information is captured by applying the least upper bound operation to a pair comprising newly received information and previously transmitted data.

Let the client process compute

$$h\Big(\bigsqcup\big(rng(ssf) \cup rng(ssg)\big)\Big) \tag{1}$$

– where $rng$ returns the set of elements in its sequence argument. Suppose that $i$ and $j$ pieces of information are available from $ssf$ and $ssg$, respectively. Then the client may compute an approximation to (1) using:

$$h\Big(\bigsqcup\{ssf(1), \ldots, ssf(i)\ ,\ ssg(1), \ldots, ssg(j)\}\Big)$$

It is necessary that $h$ be monotonic to ensure that an increase in the quality of the input approximations improves the quality of the result generated by the client.

The entire client server system is:

**Program** *web_computation*
**initially** $f, g, p, z = \{\}, \{\}, \perp, \perp$
**assign**

$$\begin{array}{llll} & \{\textit{send } f\} & ssf, f := tail(ssf), f \cup \{head(ssf)\} & \text{if } ssf \neq null \\ [] & \{\textit{send } g\} & ssg, g := tail(ssg), g \cup \{head(ssg)\} & \text{if } ssg \neq null \\ [] & \{\textit{client receive}\} & p, f, g := \bigsqcup(p, f \cup g), \{\}, \{\} & \\ [] & \{\textit{client apply}\} & z := h(p) & \end{array}$$

**end** {*web_computation*}

The system may make progress in three ways:

- a server may make available more information;
- the client may read from $f$ and $g$; or
- the client may improve its approximation to the required result (1).

The client-server system may be implemented using Grab and Go communication [7] whereby many messages (or none) may be read from an asynchronously modified port. Note that the port extraction statement

$$p, f, g := \bigsqcup(p, f \cup g), \{\}, \{\}$$

is unconditional. Thus, execution of this action need not be delayed while awaiting input – the port is processed immediately whatever its status (empty, having received one input, or having received many inputs).

A Java simulation [9] of web-based computation has been developed. The system [14] (see Figs 1, 2) can be executed from `http://how.to/grabandgo`. Partial orders are represented abstractly by an interface `Cpoable`. Ports retain only the best available information: port output, $p.write(e)$, is implemented as $p := p \sqcup e$ and port input, $p.read(x)$, is implemented as $x := p \sqcup x$. A port $p(D, \bot, \sqsubseteq, \sqcup)$ is parametrized by the data type, $D$.

# 3 Web based integration

Consider a system (see Fig 2) which computes and displays approximations to the integral $\int_a^c f(x)dx$ using the property

$$\int_a^c f(x)dx = \int_a^b f(x)dx + \int_b^c f(x)dx \text{ where } a < b < c$$

Let $a$ and $b$ be two real numbers with $a < b$, and let $n$ be an integer where $n > 0$. Define $h = |b - a|/n$ and $x_i = a + ih$. Then $\int_a^b f(x)\ dx \approx trapezoidal(f, a, b, n)$ [2] where

$$trapezoidal(f, a, b, n) = \frac{h}{2}\Big[f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)\Big]$$

Given that $f$ is increasing and convex a sequence of approximations generated by the trapezoidal rule are ordered according to:

$$n \leq n' \Rightarrow trapezoidal(f, a, b, n) \sqsubseteq trapezoidal(f, a, b, n')$$

The least upper bound of two approximations is defined, for increasing and convex $f$, as:

$$trapezoidal(f, a, b, n) \sqcup trapezoidal(f, a, b, n')) = trapezoidal(f, a, b, max(n, n'))$$

A system *integral* which generates approximations to $\int_a^c f(x)dx$ is defined below:

**Program** *integral*
  **initially**
    $i, j, best = 0, 0, 0$
    $firstPart, secondPart = \{\}, \{\}$
    $portFirstPart, portSecondPart = \bot, \bot$
  **assign**
      $i, firstPart := i + 1, firstPart \cup \{trapezoidal(f, a, b, 2^i)\}$
    $[\!]$ $j, secondPart := j + 1, secondPart \cup \{trapezoidal(f, b, c, 2^j)\}$
    $[\!]$ $portFirstPart, firstPart := \bigsqcup\big(portFirstPart, firstPart\big), \{\}$
    $[\!]$ $portSecondPart, secondPart := \bigsqcup\big(portSecondPart, secondPart\big), \{\}$
    $[\!]$ $best := portFirstPart + portSecondPart$
**end** $\{integral\}$

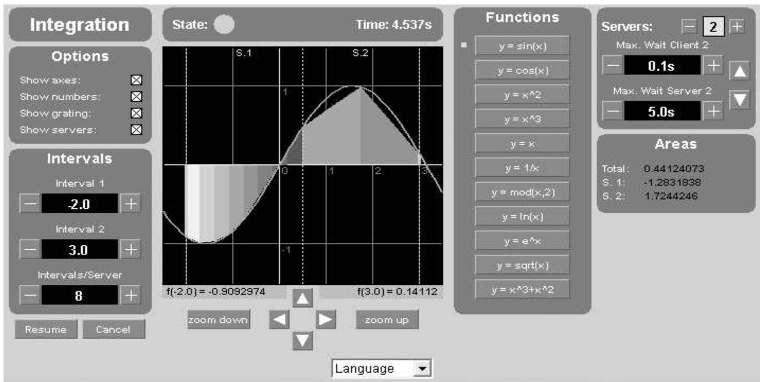


Figure 2: Two servers compute approximations to disjoint parts of an integral.

# 4 Co-algorithms

It is widely recognised that the *grid computing paradigm* can give rise to radically different ways of organising computations. Algorithms which give poor performance on conventional architectures may be reorganised for efficient implementation on computational grids. The concept of grid computing makes traditional measures of parallel efficiency to some extent redundant. The view taken here is that computation need no longer be constrained to utilise a particular machine efficiently – rather it may exploit grid resources extravagantly in order to reduce computation time. Consider a situation in which a problem may be solved by either of two iterative algorithms repeatedly applying functions $f$ and $g$. The convergence characteristics of the algorithms may be data dependent. Thus it is not possible to say, in general, which algorithm has the better convergence behaviour. Further, one algorithm may converge at a uniform rate whereas the other may initially converge very slowly but, when a good approximate solution has been found, may converge rapidly. It may be profitable to execute the algorithms concurrently, allowing them to share information from time to time with the goal of improving overall performance. Suppose that $f$ generates a series of improving approximations to a desired solution:

$$\perp \sqsubseteq f(\perp) \sqsubseteq f^2(\perp) \sqsubseteq \cdots \sqsubseteq f^k(\perp).$$

where $f^i(\perp)$ is the approximation generated on the $i$th iterative cycle, $f$ is a monotonic function and $\perp$ is an initial approximation. Suppose that $g$ possesses the same properties. It may be useful to allow the processes applying the functions $f$ and $g$ periodically to exchange information in order to improve overall performance. Such a combination of processes is called a $co - algorithm$. A UNITY template for a co-algorithm comprising processes which apply $f$ and $g$, is:

$$
\begin{array}{l}
\textbf{Program}\ \ \mathit{co\_algorithm} \\
\quad \textbf{initially}\ \ x, y, best = \perp, \perp, \perp \\
\quad \textbf{assign} \\
\qquad\qquad x := f(x) \ \ [\!]\ \ best := best \sqcup x \ \ [\!]\ \ x := x \sqcup best \\
\qquad [\!]\ \ y := g(y) \ \ [\!]\ \ best := best \sqcup y \ \ [\!]\ \ y := y \sqcup best \\
\textbf{end}\ \{\mathit{co\_algorithm}\}
\end{array}
$$

Here the iterative processes which apply $f$ and $g$ may continue in isolation until they reach their (common) fixed–point. Alternatively, each process may update or read an approximation to the best available data, *best*. Thus, one process may utilise another's approximate solution, offering the possibility of achieving the common fixed point more rapidly. The system can be extended in obvious ways to incorporate more than two iterative functions.

A simple concrete co-algorithm to compute a minimum spanning tree is developed below. The purpose of the exercise is to demonstate that, for a well understood problem, intermediate approximations generated by components of the system may differ in character from approximations generated by independently executing non-interfering processes.

**Example** A solution to the minimum spanning tree problem may be generated by using either Kruskal's or Prim's algorithm. In order to ensure that both algorithms are generating the same solution it is necessary to assume that the weights of edges of an input graph, $G$, are strictly positive and unique. In such cases the minimum spanning tree $T_G$ of $G$ is unique. A co-algorithm which combines these algorithms and which periodically shares partial solutions generated by them is:

**Program** *min_spanning*
  **initially** $k, p = \{\}, \{n\}$
  **assign**
    $k := KruskalStep(k, G) \quad [\!] \quad k := k \cup p$
  $[\!]$ $p := PrimStep(p, G) \quad [\!] \quad p := p \cup k$
**end** $\{min\_spanning\}$

Here the approximation in $k$ ($Kruskal$) need not contain only (acyclic) edges of least weight nor need the approximation in $p$ ($Prim$) be a tree. For a random graph the edges in a partial solution generated by Kruskal's algorithm are distributed "uniformly" over the whole input graph $G$. This property is not enjoyed by the partial solutions generated by Prim's algorithm. Thus, in general, the two algorithms compute $different$ approximations to $T_G$. Combining partial solutions generated by Kruskal's and Prim's algorithms is a valid operation since there is a unique minimum spanning tree and

$$edges(F) \subset edges(T_G) \wedge edges(T) \subset edges(T_G) \Rightarrow edges(F \cup T) \subseteq edges(T_G)$$

This co-algorithm could be extended by including multiple copies of Prim's algorithm. For example, consider a graph $G$ with $n$ nodes. A co-algorithm may comprise Kruskal's algorithm and $n$ instances of Prim's algorithm, each with a different start node in $G$.

# 5 Eigensystem co-algorithm

Consider a co-algorithm to determine the $m$ ($\leq n$) dominant eigenpairs of a real symmetric matrix $A \in \Re^{n \times n}$. An eigenvalue-eigenvector pair $(\lambda, x)$ of $A$ satisfies $Ax = \lambda x$. Computing subsets of eigenvalues and eigenvectors of very large, often sparse, matrices is an essential feature of many important applications. The only feasible way to perform this task is to use an iterative method in which the matrix of interest, $A$, is used solely as a multiplier. A number of independent iterative algorithms are available for the solution of the problem. The co-algorithm proposed here is constructed from a selection of these and includes an extended power method and a number of subspace iteration methods [6]. For all of these algorithms it is assumed that the eigenpairs are extracted in order of decreasing eigenvalue magnitude. This assumption simplifies the treatment of communication ports below. The extended power method below computes the eigenpairs one at a time; when an eigenpair has been isolated the method is restarted with an initial vector which is orthogonal to previously computed eigenvectors. A sequential version of the algorithm is:

**Program** *Extended_Power*
**initially** $x(1), k = e_1, 1$
**assign**

$$
\begin{array}{lll}
 & x(k) := g(x(k), A) & \text{if } \neg converged(x(k)) \\
[\!] & x(k+1), k := ortho(e_{k+1}, x(1), \ldots, x(k)), k+1 & \text{if } converged(x(k)) \wedge k < m
\end{array}
$$

**end** {*Extended_Power*}

Here $ortho(x,\ X)$ orthogonalises a vector $x$ with respect to the set of vectors $X$, $e_i$ denotes the unit vector of length $n$ whose $i$th component is 1, $converged(x)$ is true if $x$ is a sufficiently accurate approximation to an eigenvector of $A$ and $g$ denotes the iterative body of the extended power method – which includes the orthogonalisation of eigenvector approximations with respect to already converged eigenvectors.

A second method – subspace iteration – repeatedly refines $m$ approximations to the required eigenvectors using an iterative function, $h$. Each iteration of this method is more computationally intensive than an iteration of the extended power method. On each iteration approximations to *all* of the required $m$ eigenpairs are available. The eigenvector approximations are modifed according to $y(1), \ldots, y(k) := h(y(1), \ldots, y(k), A)$.

There are many ways in which to construct co-algorithms based on the methods above and their variants. Other co-algorithms may be built by including, for example, variants of the Lanczos algorithm [13]. Co-algorithms use partial eigenpair information to improve the overall convergence rate. Here, only a simple instance is presented in which information generated by a *single* subspace iteration method and the extended power method is shared. An appropriate CPO for ordering vectors can be constructed using a metric $\mathbf{M}$ given by $\mathbf{M}(x) =_{def} \| (Ax - \overline{\lambda}x) \|_2$. Here the eigenvalue approximation $\overline{\lambda}$ is given by $\overline{\lambda} = x^T Ax$ and the norm $\| x \|_2$ is defined as $\sqrt{x^T x}$. The vector approximations $x$ and $y$ are ordered according to $x \sqsubseteq y$ iff $\mathbf{M}(x) \geq \mathbf{M}(y)$. The least upper bound of two vectors $x$ and $y$ is given by:

$$
x \sqcup y =_{def} \begin{cases} x & \text{if } \mathbf{M}(x) \leq \mathbf{M}(y) \\ y & \text{otherwise} \end{cases}
$$

Then a co-algorithm to compute the $m$ dominant eigenvectors of $A$ is:

**Program** *Extended_Power_Subspace*
**initially** $x(1), k, y(1), \ldots, y(m) = e_1, 1, e_1, \ldots, e_m$
**assign**

$$
\begin{array}{lll}
 & y(1), \ldots, y(m) := h(y(1), \ldots, y(m), A) & \\
[\!] & best(1), \ldots, best(m) := best(1) \sqcup y(1), \ldots, best(m) \sqcup y(m) & \\
[\!] & y(1), \ldots, y(m) := y(1) \sqcup best(1), \ldots, y(m) \sqcup best(m) & \\
[\!] & x(k) := g(x(k), A) & \text{if } \neg converged(x(k)) \\
[\!] & best(k) := best(k) \sqcup x(k) & \\
[\!] & x(k) := x(k) \sqcup best(k) & \\
[\!] & x(k+1), k := ortho(best(k+1), x(1), \ldots, x(k)), k+1 & \\
 & & \text{if } converged(x(k)) \wedge k < m
\end{array}
$$

**end** {*Extended_Power_Subspace*}

Algorithm interaction can be restricted by limiting the non-determinism in the co-algorithm above: for example, the best available solution may be updated only when an eigenvector has converged to the required accuracy.

## 6 Results

Some preliminary experimental results are presented for a co-algorithm for computing the first $m$ dominant eigenpairs of a real symmetric matrix. The co-algorithm used comprises the following components:

- an extended power method which computes the $m$ dominant eigenvectors *one at a time* in order of decreasing eigenvalue magnitude;
- six variants of subspace iteration each of which uses a different approach to improving *simultaneously* $m$ eigenvector approximations.

The co-algorithm has been implemented in FORTRAN 90 and uses the MPI communication interface. It has been executed on an IBM SP2 with a single processor node allocated to each of its constituent algorithms. The main points of interest of the implementation are:

- A client/server model is used. Each algorithm is a client. The server acts as the ports.
- The $p$th port is associated with the $p$th eigenvector. Only the least upper bound of all received approximations and its accuracy are stored.
- Each client retains a local copy of port data;
- The system employs blocking *send*s to transfer metrics and control signals. Eigenvector transmission utilises non-blocking communication.
- A *send* inspects the accuracy of the local buffer before querying the server; data is transferred only if the information being sent is superior to the best port data.
- A *receive* inspects its local buffer first before querying the server. If the server holds more accurate information, a non-blocking *receive* is initiated.
- Non-blocking transfers use buffers. A local buffer is updated with any completed receive transfers at the beginning of each receive call. A receive transfer initiated on one call can only be used in subsequent calls to the function.
- To reduce communication the clients and server are restricted at any one instant to conducting one non-blocking send and one non-blocking receive for a given port.

Eigenpairs are checked for convergence one at a time, beginning with the dominant pair. A *send* is issued only when an eigenpair has converged. A *receive* is issued for the most dominant non-converged eigenvector after a fixed number of iterations have been performed. The system has been tested using a number of sparse matrices (selected from Matrix Market [10] or constructed from an example taken from the literature [4]). In Table 1 execution times are presented for (i) algorithms operating in isolation and (ii) a co-algorithm in which its constituent algorithms exchange eigenvector approximations. The matrices used are real symmetric, sparse of order $n$ with $nz$ non–zero elements; $m$ eigenpairs are sought. $TimeI$ and $TimeC$ are the execution times for the fastest single algorithm and the co–algorithm respectively. $UP$ denotes the number of updates made to a client's work area (beneficial receives); $CI$ denotes the ratio of the execution time of the co–algorithm to that of the fastest single algorithm.

These preliminary experimental results indicate that:

| Matrix name | $n$ | $nz$ | $m$ | $TimeI$ | $TimeC$ | $UP$ | $CI$ |
|---|---|---|---|---|---|---|---|
| BCSSTK09 | 1083 | 9760 | 4 | 10.4 | 9.6 | 2 | .93 |
| BCSSTK25 | 15439 | 133840 | 4 | 177.7 | 177.9 | 0 | 1.00 |
| 12Jennings | $10^5$ | $15 \times 10^6$ | 11 | 946 | 880 | 3 | .93 |
| 12Jennings | $10^5$ | $45 \times 10^6$ | 11 | 2030 | 1833 | 4 | .9 |
| 12Jennings | $10^6$ | $9 \times 10^4$ | 11 | 3689 | 3318 | 4 | .9 |
| 12Jennings | $10^6$ | $10^8$ | 11 | 7061 | 6538 | 3 | .93 |
| 12Gear | $10^5$ | $10^5$ | 8 | 465 | 465 | 0 | 1 |
| salyr1 | 238 | 11280 | 50 | 1098 | 50 | 36 | .045 |

Table 1: Co–algorithm Results (times in secs)

- in the cases of BCSSTK25 and 12Gear no beneficial communications are made to the fastest algorithm. However, the performance of the co-algorithm system coincides with that of the fastest individual algorithms.
- the co–algorithm performs particularly well in the case of Salyr1. In this case all of the *individual* algorithms converged slowly – with some failing to converge within the allocated time. The constituent algorithms benefit greatly from sharing information.
- for the other examples the gains are more modest with the co–algorithm outperforming the best single algorithm by, on average, 8%.

## 7 Future Work

Constraint satisfaction problem (CSP) involves computing with *partial information* using *propagate-and-search* [12] techniques – for example, $25 \leq X \leq 100$ and $0 \leq Y \leq 75$ is a constraint on the values of $X$ and $Y$. Partial information may be processed by *local deduction* to generate improved approximations. In the example above the addition of the clause $X \leq Y$ allows the deduction $25 \leq Y \leq 75$. Connections with our approach remain unexplored.

Misra [11] has shown that commutative systems can be realised as (distributed) loosely coupled systems. In general, the actions of the web and co-algorithm templates are *not* commutative. For example, the actions of the web system which update the shared variables $f$ and $p$

$$\{\mathbf{send}\}ssf, f := tail(ssf), f \cup \{head(ssf)\} \quad \text{if} \quad ssf \neq null$$
$$\{\mathbf{receive}\}p, f, g := \bigsqcup(p, f \cup g), \{\}, \{\}$$

do not commute. In one case $head(ssf)$ is first sent and is then read by $p$; in the other $p$ is updated before $head(ssf)$ is sent. However, although the web and co-algorithm templates are non-commutative they are still amenable to implementation on distributed message passing architectures. For example, although the actions above do not commute one (*receive* followed by *send*) is a subcomputation of the other in the sense that:

$$(send \ ; \ receive) = (receive \ ; \ send) \ ; \ receive$$

Thus (*receive* ; *send*) can progress, after one further action, *receive*, to the state realised by (*send* ; *receive*). It may be possible, therefore, to broaden the definition of loosely coupled systems to include shared variables which are accessed only by Grab and Go communications.

The preliminary experimental results suggest that the practical realisation of Grab and Go systems merits further investigation. Further work is in progress to determine the practicability of using partial eigensystem co-algorithms. In particular, the investigation is being broadened in three respects:

1. more algorithms (for example Lanczos variants) are being incorporated in the co-algorithm system;

2. alternative interference strategies are being investigated (the current system only sends eigenvector approximations on convergence); and

3. experimental results on a range of architectures are being compiled.

**Acknowledgement**

The authors are grateful to the referees whose comments have led to improvements in this paper and to C.A.R. Hoare for his remarks on a draft of an earlier paper on Grab and Go communication.

## References

[1] Bowman H.; Derrick, J.: *Formal Methods for Distributed Processing, a Survey of Object-Oriented approaches*, Cambridge, 2001.

[2] Burdern R., Faires, D.: *Numerical Analysis*, sixth edition, Brooks/Cole, 1997.

[3] Chandy K.M., Misra J.: *Parallel Program Design: a Foundation*, Addison-Wesley, 1988.

[4] Clint M., Jennings A.: The evaluation of eigenvalues and eigenvectors of real symmetric matrices by simultaneous iteration, Comp. Jnl., 13, 76–80, 1970.

[5] Davey B.A.; Priestley H.A.: *Introduction to Lattices and Order*, second edition, Cambridge, 2002.

[6] Demmel J. W.: *Applied Numerical Linear Algebra*, SIAM, 1997.

[7] Gabarró J., Stewart A., Clint M.: Grab and Go systems: a CPO approach to concurrent web– and grid–based computation, Foundations of wide area network computing (F-WAN), ENTCS Vol. 66.3, 2002.

[8] Gabarró J., Stewart A., Clint M.: The iterative solution of large–scale problems in numerical linear algebra: a grid–oriented Grab and Go model, PMAA'02, Neuchâtel Switzerland, Abstracts, 29, 2002.

[9] Magee, J.; Kramer, J.: *Concurrency, State Models and Java Programs*, Wiley, 1999.

[10] Matrix Market available at: http://math.nist.gov/MatrixMarket/.

[11] Misra J.: Loosely Coupled Processes, Future Generations Computer Systems 8, 269–286, North-Holland, 1992.

[12] Roy, P.V.; Haridi,S.: *Concepts, Techniques, and Models of Computer Programming.* To appear, `http://www.sics.se/~seif`.

[13] Szularz M., Weston J., Clint M.: Explicitly restarted Lanczos algorithms in an MPP environment, Parallel Computing, Vol 25, No. 5, 613–631, 1999.

[14] Vallejo I.: Visualización de sistemas Grab-And-Go (in Spanish), Final Studies Project, UPC, (to appear).