Grab and Go Systems: a CPO approach to concurrent web and grid-based computation

J. Gabarró¹ and A. Stewart² and M. Clint²

¹ Dep. LSI, Universitat Politècnica de Catalunya, Jordi Girona, 1-3, Barcelona 08034, Spain.
   gabarro@lsi.upc.es

² School of Computer Science, The Queen’s University of Belfast
   Belfast BT7 1NN, Northern Ireland, UK.
   {A.Stewart, M.Clint}@qub.ac.uk

Abstract

A useful property of web-based information systems [1] is the ability to display partial information. For example, a web program can display a sequence of fuzzy images which is extended by the production of improved images as execution of the program proceeds. A sequence of improving approximations to an image can be modelled by the elements of a complete partial order (CPO). CPOs can also be used to model grid-based computations. For example, consider a collection of iterative processes, each with identical functionality, which individually generate a series of improving approximations towards a desired goal. Sharing approximations from time to time among the processes may produce faster convergence than could be achieved by any of the processes separately. Each process in the collection has the potential to exchange partial information with its companions so that all may make use of the best information available.

In this paper a non-blocking communication abstraction, based on CPOs, is used to develop a model of iterative web- and grid-based computations. The abstraction is novel in that it may not directly match a send communication in one process with a corresponding receive communication in another; rather, a receive communication is identified with taking the least upper bound of the set of messages available at an input port - this set may be empty, contain exactly one message or contain multiple messages. In all cases the receiver does not wait but gathers the best available information and proceeds with its computation. Thus, the abstraction corresponds to a loosely coupled model of distributed computation. The applicability of the model is illustrated by a number of disparate examples of distributed iterative computation.

Keywords web computation; grid computation; Grab and Go system; CPOs.

©2002 Published by Elsevier Science B. V. Open access under CC BY-NC-ND license.
1 Introduction

The notion of distributed computation has been modified as a result of recent developments in technology - thus, traditional models of distributed computation may not be the most appropriate to represent computations realised on web servers and computational grids. An essential difference between web- and grid-based computing and conventional distributed models is in the treatment of communication transfer rates. On a single machine communication time can be accurately predicted and it is normal for input communications either to receive a message if it is available or wait until the message arrives. In contrast, there may be wide variations in delivery time for web-based communications. In this paper an alternative non-blocking form of communication based on CPOs is proposed for iterative processes each of which generates a series of steadily improving information approximations. The overall computation may be accelerated by sharing, at certain points, the best available information over all of the processes.

Suppose that an iterative process $S_{out}$ generates a series of improving approximations:

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

where $f^i(\perp)$ is the approximation generated by $S_{out}$ on its $i$th iterative cycle, $f$ is a monotonic function and $\perp$ is an initial approximation. After each approximation is generated it is sent asynchronously to another process $S_{in}$. For web- and grid-based computing it is desirable to decouple the system $S_{out} \parallel S_{in}$ as far as possible. To this end it is desirable to allow the generator process to be in a different iterative cycle from the receiver process. In a loosely coupled system the following situations may arise when the input port is inspected by $S_{in}$:

(i) no information is available;
(ii) one piece of information is available; or
(iii) multiple pieces of information are available.

A new receive operation, \textit{best.receive} is proposed which is defined for each of the cases above as follows:

(i) $S_{in}$ does not wait but accepts $\perp$;
(ii) $S_{in}$ immediately accepts the only available datum; and
(iii) $S_{in}$ immediately accepts the least upper bound ($\bigvee$) of the available approximations.

\* Work partially supported by: (1) The British Council and Spanish Ministry of Science and Technology ACI HB1999-0093, (2) Spanish CICYT under grant TIC2002-04498-C05-03 (Tracer), (3) IST program of the EU under contract IST-2001-33116 (Flags), (4) Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).
Fig. 1. *ServerOne* and *ServerTwo* send information to the *Client*. Shaded parts of the CPOs in the servers represent those parts of the CPO extracted by the client. The utilization of the information is realised by a function $f$ to generate an approximation $y$ viz $y = f(\Delta_1(x_1), \Delta_2(x_2), \Delta_3(x_3))$. Communication ports are represented by bullets. White areas in the client CPO represent local information.

In this simple situation $\bigcup\{f^i(\bot) \mid 0 \leq i \leq k\} = f^k(\bot)$. The situation may be extended to the case where there are multiple information generators - and so the least upper bound operation may have a more complex realisation. Alternatively, a generator may produce independent pieces of information which are not ordered. The novel feature of the communication is that the non-blocking *best receive* instruction may be matched with none, one or multiple *send* instructions. Two concrete examples are given below.

Consider the problem of a client computer displaying and processing image information stored on a server. At a given instant the client may only have received partial information from the server. Nevertheless, it is desirable that this partial information be displayed and, perhaps, processed by a user. This simple situation can be modelled by a single chain of approximations as indicated above. Here $f^i(\bot) \sqsubseteq f^{i+1}(\bot)$ means that the $i$th approximation to the image, $f^i(\bot)$, is a subimage of the $(i+1)$th approximation, $f^{i+1}(\bot)$. When the client receives new input it can be compared against the available data and, if the new approximation is an improvement, the new image is displayed. This example can be easily extended to cover the situation where a video image is broadcast by a web server.

The web example can be extended to the case where a *Client* process has to compute approximations from information selected from several servers (see Figure 1). Let $x^i_j$ be the $i$th approximation in chain $x_j$, $1 \leq j \leq 3$ where a chain is a sequence of improving approximations. Suppose that the chains, $x_j$, are generated by remote servers and that a local client process wishes to process parts of the information supplied by these servers. Let $\Delta_j$ be an extraction function which can be applied to an element from chain $x_j$. Then
the client may generate an approximation to:

\[ f(\Delta_1(\lim_{i \to \infty} x_i^1), \Delta_2(\lim_{i \to \infty} x_i^2), \Delta_3(\lim_{i \to \infty} x_i^3)) \]

from chains of approximations \( x_j, 1 \leq j \leq 3 \) provided that the extractor functions are Scott continuous [13] – that is

\[ \Delta_j(\lim_{i \to \infty} x_i^j) = \lim_{i \to \infty} (\Delta_j(x_i^j)), 1 \leq j \leq 3 \]

At any instant the server processes may have available only approximations, \( x_1^k, x_2^k, x_3^k \), to the three CPO limits. \( f \) is required to be monotonic in all its arguments in order to ensure that an increase in the quality of the input approximations increases the quality of the client approximation.

It is widely recognised that the concept of grid based computation may result in radically different ways of organising computations. Inefficient algorithms for traditional architectures may be appropriate for implementation on computational grids. Consider a situation in which a problem may be solved by any one of three iterative algorithms. The convergence characteristics of the various algorithms may be data dependent, and so, it is not possible to say, in general, which algorithm has the best convergence behaviour. Further, one algorithm may converge uniformly while another may initially converge very slowly but, once a good approximate solution is found, may converge rapidly. It may be profitable for the algorithms to share information from time to time in order to improve overall performance. Suppose that the generated solutions are tuples of real numbers. Let \( S_1 \) generate approximations \( \bigsqcup \{ f^i(\perp, \ldots, \perp) | 0 \leq i \} \), \( S_2 \) generate approximations \( \bigsqcup \{ g^i(\perp, \ldots, \perp) | 0 \leq i \} \) and \( S_3 \) generate approximations \( \bigsqcup \{ h^i(\perp, \ldots, \perp) | 0 \leq i \} \). At a given stage (or stages) in the process one algorithm may send an approximate solution to another algorithm. Let \( S_1 \) be on its \( i \)th iterative cycle, \( S_2 \) on its \( j \)th iterative cycle and \( S_3 \) on its \( k \)th iterative cycle when both \( S_1 \) and \( S_2 \) send information to \( S_3 \). Then, assuming no further information transfer, `best_receive` executed by \( S_3 \) accepts \( \bigsqcup \{ f^i(\perp, \ldots, \perp), g^j(\perp, \ldots, \perp) \} \). Here,

\[ \bigsqcup \{(x_1, \ldots, x_n), (y_1, \ldots, y_n)\} = (\bigsqcup \{x_1, y_1\}, \ldots, \bigsqcup \{x_n, y_n\}) \]

A receiver may choose to ignore this approximation if it has a better one available locally; alternatively, the receiver may use the new partial information in its computation - perhaps by combining the best parts of its own data with the best parts of the other Grab and Go system processes’ data before proceeding with its computation. Thus, the best information available to \( S_3 \) is

\[ \bigsqcup \{ f^i(\perp, \ldots, \perp), g^j(\perp, \ldots, \perp), h^k(\perp, \ldots, \perp) \} \]

Both the web-based and grid-based examples require the processing of partial information. It is possible to model such computations by defining an information ordering and then requiring that computations are monotonic with respect to this order. In particular, the situations above can be modelled by representing communication ports by CPOs. This leads to the notion that a `receive` communication instruction does not wait for data but processes the
least upper bound of the data available on the port - in the worst case a receive instruction returns the bottom element of the CPO (no data).

An abstract model of Grab and Go systems is developed in §2. Here definitions of a port, a loosely coupled parallel composition, $\parallel$, and the operators $send$ and $best\_receive$ are given. The abstract model is illustrated by a number of concrete examples in §3 – these include web systems and Grab and Go systems suitable for grid implementation. A general discussion of the applicability of approximation systems is given in §4.

2 Ports as CPOs

Asynchronous communications between members of a set of processes is represented by a port abstraction. A port is a shared data space. A port has an associated data structure – usually this is a FIFO queue so that the receiver can directly match inputs with outputs. Conventionally, a receive instruction applied to a non-empty port strips away the first-in datum and assigns it to a specified local variable. In contrast, Grab and Go communications are modelled by a port which retains only the best available information. Let $(D, \sqsubseteq, \sqcup)$ be a CPO [12] with domain $D$, ordering relation $\sqsubseteq$ and least upper bound operator $\sqcup$. Then $p : (D, \sqsubseteq, \sqcup)$ is a port of type $D$. The definitions of the port communication instructions $send$ and $best\_receive$ are:

1. $send(e, p) = df \quad p := p \sqcup \{ p, e \}$
2. $best\_receive(x, p) = df \quad x := p \sqcup \{ p, x \}$

Here $e$ is an expression and $x$ is a variable name. Clearly the values of $x$ and $p$ are either improved or unchanged by port operations.

$$p \sqsubseteq send(e, p)_p$$
$$x \sqsubseteq best\_receive(x, p)_x$$

Here $\sigma_z$ denotes the value of the variable $z$ in the state $\sigma$. Further, sending $\bot$ (the least element of the CPO) has no effect:

$send(\bot, p) = \Pi$

where $\Pi$ is the empty statement (SKIP). Similarly,

$$p = \bot \Rightarrow best\_receive(x, p) = \Pi$$

A system $S$ utilising a port $p : (D, \sqsubseteq, \sqcup)$ may be defined as:

$$\begin{align*}
\text{var} & \quad p : (D, \sqsubseteq, \sqcup) := \bot; \\
\text{end} & \quad p
\end{align*}$$

Here all communications through $p$ are hidden from the environment. It is assumed that $send(e, p)$ satisfies $e \in D$ and $best\_receive(x, p)$ satisfies $x \in D$. Note that a port may be initialised to a value in a CPO that is not its bottom element.
A model of the communication behaviour of a Grab and Go system can be devised by equating a multi-process system with a uni-processor implementation. The order in which the uni-processor executes instructions is not deterministic. For example, the system \( \text{send}(e,p) \parallel \text{best\_receive}(x,p) \) – where \( \parallel \) is the composition operator for systems with Grab and Go communications – may be modelled by executing the \( \text{send} \) instruction first; alternatively, the system can be modelled, with equal validity, by executing the \( \text{best\_receive} \) instruction first. The interleaving model is complicated by the asynchronous nature of \( \text{send} \) communications. An instruction \( \text{send}(e,p) \) may not immediately update port \( p \) - a delay of unknown duration may occur before the port is updated. Thus, unlike CSP, a \( \text{send} \) is not regarded as an atomic action without duration. A process cannot be identified with a single execution sequence; rather, a process may give rise to multiple execution sequences, each of which corresponds to a particular sequence of port update delays. An abstract model of the interface between a port and its associated producer and consumer processes is now devised for \textit{finite straight-line} programs.

### 2.1 Communications to ports

The aim of this section is to investigate the behaviour of communications sent to ports. A process \( S \) is deemed to have, for each of its ports, an associated set of communications, \( S \uparrow p \):

\[
S \uparrow p = \text{def} \{ a | \exists S_1, S_2. S = S_1; \text{send}(a,p); S_2 \}
\]

Using a set to represent communications means that one communication may overtake another thereby modelling a network which delivers asynchronous communications in a manner that cannot be determined \textit{a priori}. The set of communications associated with a given port \( p \) in system \( S_1 \parallel S_2 \) is:

\[
(S_1 \parallel S_2) \uparrow p = \text{def} (S_1 \uparrow p) \cup (S_2 \uparrow p)
\]

A communication set \textit{may} give rise to various sequences of port updates. Here, the trace notation of CSP [7,9] is modified in order to define such sequences. In the context of approximation communications a port trace is a sequence of values stored in the port. All such traces have a least element, say, \( d \). Let \( \text{traces}(p,d,A) \) denote the set of possible behaviours of port \( p \) where the port has a least element, \( d \), and any incoming messages must be selected from the set \( A \). Here \( t_0 \) and \( t' \) are, respectively, the first element of trace \( t \) and the tail of \( t \), therefore \( t = \langle t_0 \rangle \uparrow t' \). Then,

\[
\begin{align*}
\text{traces}(p,d,\{\}) &= \text{def} \{ \langle d \rangle \} \\
\text{traces}(p,d,A) &= \text{def} \{ t | t_0 = d \land \exists a \in A : t' \in \text{traces}(p,\bigcup\{d,a\},A - \{a\}) \} \\
& \text{where } A \neq \{\}.
\end{align*}
\]

Let \#\( t \) denote the length of trace \( t \), then \( t = \langle t_0, t_1, \ldots, t_{\#t - 1} \rangle \).

**Lemma 2.1** \( \forall t \in \text{traces}(p,d,A). \text{chain}(t) \).
Port behaviour

Fig. 2. A port \( p \) with an associated consumer process. The port \( p \) receives \( \langle t_0, t_1, t_2, t_3, t_4 \rangle \in \text{traces}(p, \perp, S \uparrow p) \). When executing \textit{best\_receive}(x, p) the variable \( x \) of the consumer system takes the values given by the trace \( \langle t_0, t_0, t_1, t_1, t_3, t_3, t_4 \rangle \). Values \( t_0, t_1 \) and \( t_3 \) are read several times and \( t_2 \) is skipped.

\textbf{Proof.} \( \forall 0 \leq i \leq (#t - 2) : t_i \sqsubseteq t_{i+1} \)

< by the definition of \textit{trace}, \( t_i \sqsubseteq \bigcup \{t_i, \ldots\} = t_{i+1} > \\

\textbf{Example 2.2} Consider the following straight-line process \( S \):
\textit{send}(a, p) ; \textit{send}(b, p) ; \textit{send}(c, p). Then, \( S \uparrow p = \{a, b, c\} \) and

\[ \text{traces}(p, \perp, \{a, b, c\}) = \{ \langle \perp, a, \bigcup \{a, b\}, \bigcup \{a, b, c\} \rangle, \]
\[ \langle \perp, a, \bigcup \{a, c\}, \bigcup \{a, b, c\} \rangle, \]
\[ \langle \perp, b, \bigcup \{b, a\}, \bigcup \{a, b, c\} \rangle, \]
\[ \langle \perp, b, \bigcup \{b, c\}, \bigcup \{a, b, c\} \rangle, \]
\[ \langle \perp, c, \bigcup \{c, a\}, \bigcup \{a, b, c\} \rangle, \]
\[ \langle \perp, c, \bigcup \{c, b\}, \bigcup \{a, b, c\} \rangle \]

Each of these traces is a chain. For example,
\[ \perp \sqsubseteq a \sqsubseteq \bigcup \{a, b\} \sqsubseteq \bigcup \{a, b, c\} \]

\[ \text{\textbf{2.2 Communications from ports}} \]

A consumer in a system \( S \) may receive any trace \( t, t \in \text{traces}(p, \perp, S \uparrow p) \) from port \( p \). Such a trace is matched to the \textit{best\_receive} instructions, for port \( p \), in the consumer process. Note, however, that there is no synchronisation in the interaction between the port trace and the input sequence (see Figure 2). Thus, a trace, \( t \), representing the behaviour of a port, may give rise to the following set of input sequences of length \( i \) (representing \( i \) values read from the port using \textit{best\_receive} instructions):

\[ t \downarrow i =_{\text{def}} \{ r | \exists j : 1 \leq j \leq \#t : r_0 = t_j \wedge r' \in t[j..\#t - 1] \downarrow (i - 1) \} \]
Here \( t[j..\#t - 1] = \langle t_j, \ldots, t_{\#t-1} \rangle \) denotes the suffix of \( t \) whose first element is \( t_j \). Note that \( t \downarrow i \) allows elements of a port trace to be skipped over – this reflects the situation where the port is updated before its current data is read. Alternatively, the same element of a port trace may be read by successive \textit{best\_receive} instructions – this reflects the situation where a port is not updated between consecutive reads. Given the trace \( t = \langle t_0, t_1, \ldots, t_{\#t-1} \rangle \), the set of input sequences can be characterised, using the notation of formal languages, as \( t^*_0 t^*_1 \ldots t^*_{\#t-1} \).

\textbf{Lemma 2.3} \( \forall r \in (t \downarrow i). \text{chain}(t) \Rightarrow \text{chain}(r) \).

\textbf{Proof.} \\
\( \forall 1 \leq j < i: r_j = r_{j+1} \lor r_j = t_k \land r_{j+1} = t_l, \ l > k. \) The result follows for all chains \( t \). \( \Box \)

\textbf{Example 2.4} Consider the port trace \( \langle \bot, a, \bigcup \{a, b\} \rangle \). This trace may give rise to any of the following input sequences of length 2: \( \langle \bot, \bot \rangle, \langle \bot, a \rangle, \langle \bot, \bigcup \{a, b\} \rangle, \langle a, \bigcup \{a, b\} \rangle, \langle a, a \rangle, \bigcup \{a, \bigcup \{b\} \} \).

\section{2.3 Algebraic laws}

The behaviour of Grab and Go systems is captured by the following \textit{send} laws for straight-line processes:

\begin{enumerate}
\item \( (\text{send}(e, p) ; \text{send}(e, p)) = \text{send}(e, p) \)
\item \( \text{send}(e_1, p); \text{send}(e_2, p) = \text{send}(e_2, p); \text{send}(e_1, p) \)
\item \( \text{send}(e_1, p); \text{send}(e_2, q) = \text{send}(e_2, q); \text{send}(e_1, p) \)
\item \( x := f; \text{send}(e, p) = \text{send}(e_f, p); x := f \)
\end{enumerate}

In each case a \textit{send} instruction gives rise to non-determinism – for example,

\( \text{send}(e_1, p); \text{send}(e_2, p) = \text{send}(e_1, p); \text{send}(e_2, p) \sqcap \text{send}(e_2, p); \text{send}(e_1, p) \)

Here \( \sqcap \) denotes non-deterministic selection. Note that, in general:

\( (\text{send}(e_1, p) ; \text{send}(e_2, p)) \neq \text{send}(\bigcup \{e_1, e_2\}, p) \)

The communication sequence on the left hand side can be used to generate more execution sequences than can be constructed from the right hand side. The properties of the composition \( \text{send}(e, p) ; \text{best\_receive}(x, q) \) can be derived by considering a \textit{send} instruction to comprise separate evaluation and update actions:

\( \text{send}(e, p) =_{\text{def}} z := e; \text{update}(z, p) \)

Here \( z \) is a new variable name that cannot be used elsewhere and

\( \text{update}(z, p) =_{\text{def}} p := \bigcup \{p, z\} \)

Then,

\( \text{update}(z, p) ; S = S ; \text{update}(z, p) \)
provided that $z$ does not occur in $S$. Thus,

$\text{update}(z, p); \text{best\_receive}(x, q) = \text{best\_receive}(x, q); \text{update}(z, p)$

These rules ensure that the dependent composition $\text{best\_receive}(x, q); \text{send}(x, p)$ does not commute:

$\text{send}(x, p); \text{best\_receive}(x, q) \not\sqsubseteq \text{best\_receive}(x, q); \text{send}(x, p)$

A Grab and Go system is identified with all possible interleavings of the multiple execution sequences of its underlying processes. There is no requirement for an interleaving to synchronise input and output communications (unlike the situation in CSP where the composition operator imposes constraints, via shared actions, on the ways that the actions of processes are interleaved).

$\sqcup$ is commutative and associative:

(7) $S_1 \sqcup S_2 = S_2 \sqcup S_1$

(8) $S_1 \sqcup (S_2 \sqcup S_3) = (S_1 \sqcup S_2) \sqcup S_3$

Grab and Go communications obey the following communication law:

(9) $(\text{send}(e, p) \sqcup \text{best\_receive}(x, p)) = p, x := \bigcup \{p, e\} \cap \bigcup \{x, p, e\} \cup \{x\}$

Here both the port $p$ and the receiver’s input variable $x$ are updated. Note that the semantics of the very loose coupling operator $\sqcup$ allows the receiver to proceed (to update $x$) if the message $e$ is not immediately available.

**Example 2.5** Grab and Go systems are nondeterministic.

```
var p : (D, \sqsubseteq, \bigcup) := \bot;
send(f(\bot), p); send(f^2(\bot), p) \sqcup x := g(\bot); \text{best\_receive}(x, p)
end p
```

Assuming that $f(\bot) \sqsubseteq f^2(\bot)$, all of the following outcomes are possible for the value of $x$ in the final state:

- $x = g(\bot)$
- $x = \bigcup \{g(\bot), f(\bot)\}$
- $x = \bigcup \{g(\bot), f^2(\bot)\}$

\[\square\]

### 2.4 Deadlock and termination

Grab and Go systems cannot deadlock. Consider an empty process $\Pi$ with access to a communication port $p$. Then,

$(\text{send}(e, p) \sqcup \Pi) = \text{send}(e, p)$

$(\text{best\_receive}(x, p) \sqcup \Pi) = \text{best\_receive}(x, p)$

Again there are no requirements that an output communication be matched with an input communication.
Example 2.6 There is no requirement that the interleaved model should execute a send instruction first.

```plaintext
var p, q : (D, ⊑, ⨆) := ⊥, ⊥;
   send(f(⊥), p) ; best_receive(x, q) || send(g(⊥), q) ; best_receive(y, p)
end p, q
```

Thus, the weakest interleaving arises when all best_receive instructions are executed before the execution of any send instructions. This is the worst-case situation in which all communications are delayed until after computation has terminated and so, for the weakest interleaving, \( x = ⊥ ∧ y = ⊥ \) in the final state. □

One class of Grab and Go systems – those composed of individually terminating processes – is straightforward to reason about. Consider a system

\[
(S_1(x_1) || S_2(x_2) || \cdots || S_n(x_n))
\]

where

(i) the alphabet of each \( S_i, 1 \leq i \leq n \) is \( \alpha(S_i) \);
(ii) the input communication variable(s) of \( S_i \) is \( x_i \);
(iii) each process \( S_i, 1 \leq i \leq n \), is monotonic with respect to its input communication variables (i.e. \( x ⊑ y \Rightarrow (\forall z \in \alpha(S_i) : S_i(x)z \subseteq S_i(y)z) \)); and
(iv) each process \( S_i, 1 \leq i \leq n \), is guaranteed to terminate irrespective of its input communications.

This system may give rise to non-deterministic behaviour. Let \( ||^I \) denotes independent composition. The weakest behaviour that the system may engage in is given by

\[
(S'_1 ||^I S'_2 ||^I \cdots ||^I S'_n)
\]

where \( S'_i \) denotes the process \( S \) stripped of all send communications and with all input instructions of the form best_receive \( (x, p) \) replaced by the assignment \( x := \bigcup \{x, ⊥\} \). This new system corresponds to the worst case behaviour where all input communications arrive after all best_receive instructions have been executed. Thus, one way to reason about the original system is to consider only the worst case behaviour. The original system is guaranteed to produce the same or better information than in the worst case.

Example 2.7 The case of potentially infinite processes is more complex.

```plaintext
var p : (D, ⊑, ⨆) := ⊥;
   send(f(⊥), p) || (x := ⊥ ; while (x = ⊥) do best_receive(x, p))
end p
```

One possible interleaving of the processes above is generated solely from an infinite sequence of best_receive commands. This interleaving can be excluded by imposing a fairness constraint which necessitates that if a send instruction
is offered then it will be executed after a finite amount of time. □

Further consideration of the validity of fairness constraints is deferred until the final section.

3 Concrete Examples

The abstract model of Grab and Go systems is made concrete through a number of examples chosen from web- and grid-based computing applications.

Example 3.1 Small_Web
Consider a system Small_Web which comprises a server process generating and sending a series of approximations to an image and a browser process which receives these approximations [5]. Given a set of image co-ordinates, \( PIXEL \), and a set of colours, \( COLOUR \), then an image has type \( PIXEL \times COLOUR \). As a result of the communications the browser may be able to generate images with higher degrees of resolution.

\[
\text{Small}_\text{Web}(k : \text{int}) \equiv \\
\text{var} \; p : (\text{image}, \subseteq, \cup) := \emptyset; \\
(\text{Server}(k) \parallel \text{Browser}(k)) \\
\text{end} \; p
\]

The server program iterates \( k \) times and, on each iteration, sends a portion of the image to \( p \). Let \( \text{recover}(\text{picture}, k) \) select, according to the parameter \( k \), a series of pixel/colour pairs from the image, \( \text{picture} \):

\[
\text{Server}(k : \text{int}) \equiv \\
\text{var} \; i : \text{int}; \\
\text{for} \; i := 0 \; \text{to} \; k \; \text{do} \\
\text{send}(\text{recover}(\text{picture}, k), p) \\
\text{end} \\
\text{end} \; i
\]

The \( \text{Browser} \) periodically inspects \( p \) and, if possible, improves its current approximation to the image:

\[
\text{Browser}(k : \text{int}) \equiv \\
\text{var} \; y : \text{image}, i : \text{int} := \emptyset; \\
\text{for} \; i := 0 \; \text{to} \; k \; \text{do} \\
\text{best} \text{. receive}(y, p); \\
\text{display}(y) \\
\text{end} \\
\text{end} \; y, i
\]

Here the least upper bound operation is set union. Note that the series of approximations generated by the server may be disjoint and that the sequence of approximations stored by the port is a chain. □
Example 3.2 Bitmap Images

In a variant of the preceding example an image may be represented by bitmap approximations. Each approximation consists of the following information:

\[
\langle \text{total size} \mid \text{grey level}, \text{grey level} \ldots \rangle
\]

For instance, the image \([0, 0, 1, 0, 1, 1]\) may be represented by the following sequence of approximations (see Fig 3):

\[
\langle 8 \mid 4 \rangle \sqsubseteq \langle 8 \mid 1, 3 \rangle \sqsubseteq \langle 8 \mid 0, 1, 1, 2 \rangle \sqsubseteq \langle 8 \mid 0, 0, 0, 1, 1, 0, 1, 1 \rangle.
\]

The first approximation could be displayed by averaging the value 4 over 8 points while the second approximation could be displayed by approximating the first four points by the value 0.25 and the second 4 points by the value .75. Thus, the degree of resolution of the picture increases as the accuracy of the approximations improves. The ordering relation, \(\sqsubseteq\), is reflexive and transitive and satisfies:

\[
\langle n \mid u + v, w + x \rangle \sqsubseteq \langle n \mid u, v, \ldots, w, x \rangle
\]

Note that \(\text{bitmap}_1 \sqsubseteq \text{bitmap}_2 \Rightarrow \bigcup \{\text{bitmap}_1, \text{bitmap}_2\} = \text{bitmap}_2\)

Example 3.3 Display_Integral

Consider a system Display_Integral which computes and displays approximations to the integral \([14]\)

\[
\int_{0}^{2} f(x)dx,
\]
Fig. 4. An illustration of an approximation to $\int_0^2 f(x)dx$. Here $\int_0^1 f(x)dx$ is approximated by $L(f, C_{0,1}) = 1/2(f(0) + f(1/2))$ and $\int_1^2 f(x)dx$ is approximated by $L(f, C_{1,2}) = 1/4(f(1) + f(5/4) + f(3/4) + f(7/4))$.

using the property

$$\int_0^2 f(x)dx = \int_0^1 f(x)dx + \int_1^2 f(x)dx.$$ 

Before considering the construction of the system some preliminary definitions of intervals and integral approximations are given. $x \in \{0, 1\}^\ell$ denotes that $x$ is a binary string of length $\ell$. Let $x = x_1x_2 \ldots x_\ell$. Then, given a non-negative integer, $n$, $n.x = n.x_1x_2 \ldots x_\ell$ denotes the number $n + x_12^{-1} + x_22^{-2} + \cdots + x_\ell2^{-\ell}$.

The interval $[n \ldots n + 1]$ may be characterised by a family of partitions, $\{C_{n,\ell}| \ell \geq 0\}$ where

$$C_{n,\ell} = \{[n.x \ldots n.x + 2^{-\ell}] | x \in \{0, 1\}^\ell\},$$

For example,

$$C_{0,2} = \{[0.x \ldots 0.x + 1/4] | x \in \{0, 1\}^2\}$$

$$= \{[0.00 \ldots 0.00 + 1/4], [0.01 \ldots 0.01 + 1/4], [0.10 \ldots 0.10 + 1/4], \ldots\}$$

$$= \{[0 \ldots 1/4], [1/4 \ldots 1/2], [1/2 \ldots 3/4], \ldots\}$$

The sequence $\{\ell \mapsto C_{0,\ell}| \ell \geq 0\}$ expands to:

$$C_{0,0} = \{[0 \ldots 1]\},$$

$$C_{0,1} = \{[0 \ldots 1/2], [1/2 \ldots 1]\},$$

$$C_{0,2} = \{[0 \ldots 1/4], [1/4 \ldots 1/2], [1/2 \ldots 3/4], [3/4 \ldots 1]\},$$

$$\vdots$$

In going from $C_{0,1}$ to $C_{0,2}$ the subinterval $[0 \ldots 1/2]$ is refined into the pair $[0 \ldots 1/4], [1/4 \ldots 1/2]$. In Fig 4 there is a partition of $[0 \ldots 1]$ by $C_{0,1}$ and
a partition of \([1 \ldots 2]\) by \(C_{1,2}\). Given a continuous non-decreasing and non-negative function \(f\) over the interval \([n \ldots n+1]\) and a partition \(C_{n,\ell}\) a lower bound on the integral above is given by the Riemann lower sum (see Fig 4 for examples of Riemann sums):

\[
L(f, C_{n,\ell}) = \sum_{x \in \{0,1\}^\ell} f(n.x)2^{-\ell}.
\]

It follows that

\[
L(f, C_{n,\ell}) \leq \int_{n}^{n+1} f(x)dx,
\]

If \(\ell \leq \ell'\) then it follows that \(L(f, C_{n,\ell}) \leq L(f, C_{n,\ell'})\). A partial order for integral approximations is \((\mathbb{R}^*, \leq, \max)\) where \(\mathbb{R}^*\) denotes the non-negative reals; the bottom element of the order is 0.

The system Display_Integral consists of three processes:

\begin{verbatim}
Display_Integral(k) ≡
    var p, q : (\mathbb{R}^*, \leq, max) := 0,0,0;
    (Riemann_Sum(0, k, p) \parallel Riemann_Sum(1, k, q) \parallel Collect_Sum(k, p, q))
end p, q

Riemann_Sum(n, k) sends the first k Riemann approximations to \(\int_{n}^{n+1} f(x)dx\)
to port p:

Riemann_Sum(n, k: int, p : (\mathbb{R}^*, \leq, max)) ≡
    var i: int;
    for i := 0 to k do
        send( L(f, C_{n,i}), p);
    end
end i

Riemann_Sum sends the chain of messages:

\[
L(f, C_{n,0}) \sqsubseteq L(f, C_{n,1}) \sqsubseteq \cdots \sqsubseteq L(f, C_{n,k}).
\]

From a mathematical point of view, it is sufficient to communicate only the last value \(L(f, C_{n,k})\). However, from a computational point of view, the earlier terms in the series have less complexity and can be generated more quickly.

From a web perspective it may be advantageous to send approximate information to a client rather than waiting until an accurate solution is available.

Process Collect_Sum collects and displays two series of Riemann sums approximations from ports p and q, respectively (see Fig 4 for an example of a displayed value):
Collect_Sum\( (k: \text{ int}) \equiv \)
\[
\text{var } \text{firstHalf, secondHalf: } \mathbb{R}^*, i: \text{ int} := 0.0,0.0,0;
\]
\[
\text{for } i := 0 \text{ to } k \text{ do }
\]
\[
\text{best_receive(} \text{firstHalf}, p); \\
\text{best_receive(} \text{secondHalf}, q); \\
\text{display(} \text{firstHalf} + \text{secondHalf})
\]
\end{end};
end firstHalf, secondHalf, i

**Lemma 3.4** System Display Integral displays a chain of values taken from:
\[
\{L(f,C_{0,i}) + L(f,C_{1,j}) \mid -1 \leq i,j \leq k\},
\]
where \( C_{n,-1} = 0.0 \)

**Proof.** Riemann_Sum\( (0, k, p) \) and Riemann_Sum\( (1, k, q) \) generate the following sequences of (send) communications on ports \( p \) and \( q \) respectively:
\[
\{L(f,C_{0,i}) \mid -1 \leq i \leq k\}
\]
\[
\{L(f,C_{1,j}) \mid -1 \leq j \leq k\}
\]
These sequences may be permuted and interleaved to model the actual order of arrival of messages in the collector processor. Let \( C_{n,-1} \) denote the bottom element of the ports \( p \) and \( q \) \((0.0)\). Then the system behaviour is determined by all interleavings of the \( 2k \) best_receive instructions with the permuted and interleaved input sequences. For example, let \( v^i \) denote the value of variable \( v \) in iteration \( i \) of the collector process. Then
\[
\text{firstHalf}^0 = \cap \{L(f,C_{0,i}) \mid -1 \leq i \leq k\}
\]
and
\[
\text{secondHalf}^0 = \cap \{L(f,C_{1,j}) \mid -1 \leq j \leq k\}
\]
Further,
\[
\text{firstHalf}^0 + \text{secondHalf}^0 \sqsubseteq \cdots \sqsubseteq \text{firstHalf}^k + \text{secondHalf}^k
\]
\( \square \)

In the worst case the best_receive instructions are executed before the send instructions and so only the weakest approximation, viz 0.0, is displayed. In order to avoid this possibility, termination of the collector process could be made dependent on the value of the current integral approximation. For example, the generators could be extended to send a series of both lower and upper bounds. The collector could have a termination condition which requires that the combined lower and combined upper approximations differ by, at most, a prescribed tolerance. \( \square \)

**Example 3.5** Eigen Grab and Go System
Consider a Grab and Go system to determine the \( m, 0 < m \leq n \), dominant
eigenpairs of \( A \in \mathbb{R}^{n \times n} \). 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 important 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 Grab and Go system proposed here is constructed from two of these: Lanczos’ method and subspace iteration [4]. The idea is to compute \( m \) eigenpairs one at a time. The convergence rates of the individual eigenpairs cannot be determined in advance and it is not known whether the subspace or Lanczos algorithms will compute the next one more quickly. As they proceed each generates imprecise information about the eigenpairs yet to be computed. It may be that it would be helpful to one or other to exchange this information so that each can reap the benefits thereby achieving a faster solution that might be obtained using either of the algorithms in isolation.

For both algorithms it is assumed that the eigenpairs are extracted in order of decreasing eigenvalue magnitude. This assumption simplifies the treatment of communication ports below. In the following, the conditional statement \( S_1 \triangleleft b \triangleright S_2 \) satisfies \( S_1 \triangleleft \text{true} \triangleright S_2 = S_1 \) and \( S_1 \triangleleft \text{false} \triangleright S_2 = S_2 \). The norm \( \| x \|_2 \) is defined as \( \sqrt{x^T x} \).

**Lanczos’ method.** The variant of Lanczos’ method that is considered here computes the eigenvectors one at a time [15]; when an eigenvector has been isolated the method is restarted with a new vector (which is orthogonal to previously computed eigenvectors). A sequential version of the algorithm is:

\[
Lanczos(A; \mathbb{R}^{n \times n}, X: \text{Set of } (\mathbb{R}^n), x: \mathbb{R}^n) \equiv
\begin{cases}
\{ & \\
\quad X \cup \{x\} & \\
\quad \texttt{\#X} \cup \{x\} = m \triangleright & \\
\quad \texttt{Lanczos}(A, X \cup \{x\}, \text{ortho}(u, X \cup \{x\})) & \\
\quad \texttt{\triangleright converged}(x) \triangleright & \\
\quad \texttt{Lanczos}(A, X, f(x)) & \\
\}\end{cases}
\]

In the call \( \text{Lanczos}(A, X, x) \), the parameter \( X \) is a set of eigenvectors with \( \#X < m \) and \( x \) is an approximation to the \( \#X + 1 \) eigenvector. There are three cases:

- When \( x \) is an acceptably accurate approximation to an eigenvector and \( \#X = m - 1 \), \( x \) is the last \( m \)-th eigenvector. Thus, all of the eigenvectors have been computed and the the set \( X \cup \{x\} \) is returned.
- When \( x \) is an acceptably accurate approximation to the \( m \)-th eigenvector
but \( X < m - 1 \), more eigenvectors remain to be computed. The set of accurate eigenvectors is extended to \( X \cup \{x\} \) and the procedure is restarted with an initial approximation (usually very inaccurate) to the next eigenvector which is orthogonal to \( X \cup \{x\} \).

- When \( x \) is an inaccurate approximation to the \( m \)-th eigenvector this approximation is improved.

The procedures and functions used in \( \text{Lanczos}(A, X, x) \) are defined as follow:

- \( \text{ortho}(u, X) \) orthogonalises a vector \( u \) with respect to the set of orthogonal vectors \( X \). This can be achieved using, for example, the well-known Modified Gram-Schmidt algorithm [6]. The vector \( u \in \mathbb{R}^n \) is chosen to have each of its components 1.

- \( \text{converged}(x) \) is true if \( x \) is a sufficiently accurate approximation to an eigenvector of \( A \) and is false otherwise. Convergence is assumed when \( ||Ax - \lambda x||_2 < \epsilon \) where \( \epsilon \) is the chosen accuracy and \( \lambda \) is the eigenvalue approximation associated with \( x \).

- \( f \) denotes the iterative body of Lanczos’ method: \( f(x) \) generates from the vector \( x \), using the chosen Lanczos algorithm, a new approximation to the eigenvector of \( A \) currently being sought. This involves modifying an (expanding) matrix of so-called Lanczos vectors, \( Q_j \), generated using a characteristic Lanczos three term recurrence relation [6] by postmultiplying them with the matrix of eigenvectors of a tri-diagonal matrix given by \( Q_j^T A Q_j \) (\( Q^T \) signifies matrix transpose) and selecting from these the eigenvector approximation of interest.

**Subspace iteration method.** A second method – subspace iteration – repeatedly refines \( m \) approximations to the required eigenvectors. Each iteration of this method is more computationally intensive than an iteration of Lanczos’ method but its storage requirements are fixed. On each iteration approximations to all required \( m \) eigenpairs are available.

\[
\text{Subspace}(A: \mathbb{R}^{n \times n}, x_1 \ldots x_m : \mathbb{R}^n) \equiv \\
\{ \\
\quad x_1 \ldots x_m \\
\quad < \text{converged}(x_1 \ldots x_m) \triangleright \\
\quad \text{Subspace}(A, g(x_1 \ldots x_m)) \\
\}
\]

Here:

- \( \text{converged}(x_1, ..., x_m) \) is true if the vectors \( x_1, ..., x_m \) are acceptably accurate approximations to the dominant \( m \) eigenvectors of \( A \), and is false otherwise. Convergence is assumed when for all \( 1 \leq i \leq m, ||Ax_i - \lambda_i x_i||_2 < \epsilon \) where \( \epsilon \) is the chosen accuracy and \( \lambda_i \) is the eigenvalue approximation associated with \( x_i \).
• \( g \) denotes the iterative body of subspace iteration: \( g(x_1, \ldots, x_m) \) modifies the eigenvector approximations \( [x_1, \ldots, x_m] \), according to the chosen subspace iteration algorithm, to produce a new set of \( m \) eigenvector approximations. Essentially, the vectors \( [x_1, \ldots, x_m] \) are modified by postmultiplying them by the matrix of eigenvectors of the \( m \times m \) symmetric matrix \( [x_1, \ldots, x_m]^T \cdot A \cdot [x_1, \ldots, x_m] \) (\( [x_1, \ldots, x_n]^T \) signifies matrix transpose) and the resulting set of \( m \) vectors are orthogonalised using, for example, the Modified Gram-Schmidt algorithm.

A Grab and Go system. There are many ways in which to construct a Grab and Go system based on variants of the methods sketched above. These systems periodically use shared partial eigenpair information to improve the overall convergence rates. Here, only a simple Grab and Go system is presented which uses information generated by subspace iteration to improve the convergence rate of Lanczos’ method. A CPO for ordering vectors can be constructed using a metric \( M(x) = \| (Ax - \bar{\lambda}x) \|_2 \). Here the eigenvalue approximation \( \bar{\lambda} \) is given by \( \bar{\lambda} = x^T Ax \). The vector approximations \( x \) and \( y \) are ordered according to \( x \sqsubseteq y \) iff \( M(x) \geq M(y) \). The least upper bound of two vectors \( x \) and \( y \) is given by:

\[
\sqcup \{x, y\} =_{def} \begin{cases} 
\begin{array}{ll}
x & \text{if } M(x) \leq M(y) \\
y & \text{otherwise}
\end{array}
\end{cases}
\]

The \( i \)-th column of the unit matrix, \( I \in \mathbb{R}^{n \times n} \), is denoted by \( e_i \). Then a co-operating system which computes the \( m \) dominant eigenvectors of \( A \) is:

\[
\text{GrabAndGoEigenAlgorithms}(A; \mathbb{R}^{n \times n}) \equiv
\begin{var}
Lp_1, \ldots, Lp_m : (\mathbb{R}^n, \sqsubseteq, \sqcup) := e_1, \ldots, e_m;
\text{GrabAndGoLanczos}(A, \{\}, e_1) \parallel \text{GrabAndGoSubspace}(A, e_1, \ldots, e_m)
\end{var}
end Lp_1, \ldots, Lp_m
\]

In the Grab and Go version of subspace iteration, the best available information about the eigenvectors of \( A \) is sent to the ports \( Lp_1, \ldots, Lp_m \):

\[
\text{GrabAndGoSubspace}(A; \mathbb{R}^{n \times n}, x_1 \ldots x_m : \mathbb{R}^n) \equiv
\begin{var}
\begin{array}{c}
x_1 \ldots x_m < converged(x_1 \ldots x_m) >
\end{array}
\end{var}
\begin{var}
\begin{array}{c}
\text{for } i := 1 \text{ to } m \text{ do send}(x_i, Lp_i) \text{ end;}
\text{GrabAndGoSubspace}(A, g(x_1 \ldots x_m))
\end{array}
\end{var}
\}
\]
GrabAndGoLanczos may use port information to improve its convergence rate:

\[
\text{GrabAndGoLanczos}(A: \mathbb{R}^{n \times n}, X: \text{Set of } (\mathbb{R}^n), x : \mathbb{R}^n ) \equiv \\
\begin{cases} \\
X \cup \{x\} \\
\langle \#X \cup \{x\} = m \rangle \\
\quad \{ \\
\quad \quad X := X \cup \{x\} ; \ x := u ; \ \text{best\_receive}(x, Lp_{\#X+1}); \\
\quad \quad \text{GrabAndGoLanczos}(A, X, \text{ortho}(x, X)) \\
\quad \} \\
\langle \text{converged}(x) \rangle \\
\quad \text{best\_receive}(x, Lp_{\#X+1}); \ \text{GrabAndGoLanczos}(A, X, f(x)) \\
\} 
\]

Here \text{GrabAndGoSubSpace} is used to improve \text{GrabAndGoLanczos}' current eigenvector approximation, or, in the case of a restart, to provide an initial starting vector. The algorithm \text{GrabAndGoLanczos}(A, X, x) is adapted from \text{Lanczos}(A, X, x). As before there are three cases.

- The algorithm returns \( X \cup \{x\} \) when \( \#X = m - 1 \) and \( \text{converged}(x) \) holds (as in the case of \text{Lanczos}(A, X, x)).
- In the case where \( \#X < m - 1 \) and \( \text{converged}(x) \) is true the set \( X \) is extended to \( X \cup \{x\} \). After this update, \( \#X < m \) and the algorithm is restarted to compute the \( \#X + 1 \) eigenvector. As the initial approximation for this eigenvector \text{GrabAndGoLanczos} chooses between \( u \) (as in sequential \text{Lanczos}) and the best current approximation to the \( \#X + 1 \) eigenvector computed by \text{GrabAndGoSubSpace} and transmitted through port \( Lp_{\#X+1} \).
- When \( \text{converged}(x) \) is false, the vector \( x \) is an inaccurate approximation to the \( \#X + 1 \) eigenvector. A better approximation is the computed by applying \( f \). Before applying \( f \) \( x \) is updated to the best current \( \#X + 1 \) eigenvector approximation computed so far by \text{GrabAndGoSubSpace} and transmitted through the port \( L_{\#X+1} \).

\text{GrabAndGoEigenAlgorithms} is just one of many possible ways of combining Lanczos methods with subspace iteration methods. Many other combinations can be formulated. The Grab and Go approach provides a uniform framework within which these alternative methods may be developed.

\[\square\]

### 4 Discussion

Web-based computing is characterised by collections of participating processes sharing information over unpredictable communication networks. In the worst
case such systems must function with broken links. Information on server pages may be delivered in discrete stages and so a client process may, at times, have only partial information available. Similarly, a client process may not wait for complete information before passing on its data to a user. In order to capture the essence of web-based computing a novel loosely coupled model of distributed computing based on CPOs has been presented. The model does not impose fairness constraints – this reflects the possibility of broken links in a network. It transpires that the model which was developed to describe web-computation is also appropriate for the description of certain kinds of grid-based computations.

The advent of Grid computing will lead to a re-evaluation of how best to build applications software which can effectively exploit heterogeneous, widely dispersed resources. In many important current applications, in which heavy use is made of numerical linear algebra, the volume of data to be processed and the sparsity of the very large matrices involved points to the use of iterative methods as the only feasible approach to the solution of certain constituent problems – such as the solution of linear equations and the computation of (partial) eigensystems. There is a need to reappraise these methods in the light of the sorts of facilities which the Grid provides. In particular, the use of polyalgorithms [11] and Grab and Go systems offers the possibility of improved performance in a Grid environment. In a Grab and Go system, a collection of iterative algorithms, each having the same functionality, periodically assist each other in the solution of a problem by supplying to one another their current approximate solutions in the hope that (at least) one of them (not necessarily the sender) can exploit the best of these better than any of the others. The Grab and Go approach is modelled by the proposed port-based computational model. The approach has been illustrated by a (simple) Grab and Go system for the computation of a partial eigensolution of a real symmetric matrix.

A model of only finite straight-line processes has been presented here. Thus, a number of questions remain unresolved, including:

(i) **treatment of infinite processes**: is a fairness constraint needed to ensure that an infinite sequence of send instructions cannot, in its entirety, overtake an infinite sequence of best receive instructions in the interleaved model?

(ii) **semantics**: the semantics of Grab and Go systems is based on traces (see §2). At present work is under way to develop an alternative model based on non-deterministic actions over shared variables (UNITY [2]).

(iii) **implementation**: can an efficient implementation of the model be devised? A prototype implementation based on classes [3] is planned. At present MPI [10] supports asynchronous non-blocking send and receive instructions. The question arises as to whether efficient implementations of send and best receive can be developed from their non-blocking MPI
counterparts.

(iv) **relation to other models**: the ideas presented here might be combined with the mechanisms of concurrent constraint programming languages [8] to produce something interesting.

**Acknowledgement**

The authors are very grateful to three anonymous referees and to C.A.R. Hoare for their comments on earlier drafts of this paper.

**References**


