A Parallel Algorithm for the Global Minimization of Gibbs Free Energy

S. Berner   K.I.M. McKinnon   C. Millar

MS 96-004a

November 1996

(Revision of MS 96-004. Submitted to Annals of Operations Research)

Department of Mathematics and Statistics
University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ
Tel. (33) 131 650 5042   E-Mail : ken@maths.ed.ac.uk
http://www.maths.ed.uk/mckinnon

A Parallel Algorithm for the Global Minimization of Gibbs Free Energy

S. Berner, K.I.M. McKinnon and C. Millar
Department of Mathematics and Statistics,
University of Edinburgh EH9 3JZ, UK
E-mail: http://www.maths.ed.ac.uk/mckinnon/, sonja@math.uni-wuppertal.de

A chemical mixture under conditions of constant temperature and pressure may split into different phases. The number of phases and the composition of each may be determined by globally minimizing the Gibbs free energy of the system. This can be done by iterating between an easy local minimization problem with a high number of variables and a difficult global search and verification problem in a small number of variables. The global problem can be solved by a branch and bound method using bounds from interval analysis. When implemented in parallel, the method has lower communication requirements than other related branch and bound approaches for general global minimization. We present a parallel implementation on a network cluster of workstations that exploits this characteristic. On difficult instances, utilizations of over 90% are obtained using up to 14 processors. The algorithm copes well with varying workstation loads and has low communication overheads. A method of assessing the performance of a parallel algorithm on a shared heterogeneous network of workstations is developed.

Subject classification: AMS(MOS) 90C90, 90C99, 65Y05, 68Q22, 80-08, 80A10, 80A15

Keywords: Global optimization, interval-analysis, parallel programming, branch and bound, Gibbs free energy, phase equilibrium, primal-dual methods

1 Introduction

If a mixture of chemical substances, called the *feed*, is allowed to reach equilibrium, it may separate into different physically separated substances called *phases*. For example, at normal temperatures and pressures, a feed consisting of a mixture of oil and water separates into two phases, one of which is mainly oil and one of which is mainly water. Within a single phase the composition is constant, but different phases can have different compositions. The problem of determining the number of phases present and the amount and composition of each phase is of central importance in chemical and petroleum engineering. For example, the calculation appears in the inner loops of oil reservoir and distillation column simulators.

The commonly used techniques for the chemical and phase equilibrium problem divide into two classes [11]. The first concentrates on solving the set of nonlinear equations based on the equilibrium-constant expressions and the material-balance equations. In the second approach a local minimum of a thermodynamic function is found. Under conditions of constant temperature and pressure, the necessary thermodynamic function is the Gibbs free energy (GFE) and the problem is referred to as the *isothermal* phase equilibrium problem. Neither of the above methods are guaranteed to find the correct solution. The number of phases present in equilibrium is not known a priori, and it is often the case that there are multiple solutions to the equations in the equation solving approach and multiple local minima the minimization approach. As a result some heuristics are usually used to reduce the likelihood of these methods giving the wrong solution.

A system will be at equilibrium under conditions of constant temperature and pressure if and only if its GFE is at a *global* minimum. It follows that the number of phases and the amount and composition of each phase which occurs at equilibrium can be found by finding the global minimum of the GFE. This paper considers a method, Global Improvement Local Optimization method (GILO) [7, 6], which always finds the global minimum and which is therefore guaranteed to find the number of phases which is present at equilibrium, and the composition and amount of each phase which is present.

The global improvement (GI) step of the GILO method is computationally intensive but is well suited to parallel processing using a branch and bound approach. We have chosen to implement this GI step on a cluster of workstations linked by a local area net, as this is a commonly available resource which is often under utilized and the necessary support environment is readily available. Workstation clusters have a poor ratio of communication to CPU speed and often consist of machines of different performance. Also machines have to be shared with other users. It is therefore important that the parallel algorithm is robust against different machine speeds and changing workstation workloads caused by other users, and that it does not itself have a serious affect on the other users. These issues are addressed successfully by the method described in this paper.

The remaining sections of this paper have the following structure. In Section 2, the mathematical form of the problem is given and the GILO method used to solve it is described. The differences between the global improvement (GI) step and the general global optimization problem are also described in this section. In Section 3, the methods used to parallelize the GI step in GILO are described. In Section 4, a method of assessing the performance of a parallel algorithm on a shared heterogeneous network of workstations is developed, and the results of an implementation of the GILO algorithm on a workstation cluster are given. Conclusions follow in Section 5. The appendix gives the UNIFAC empirical model for GFE, data for the examples used in the paper and their optimal solutions.

2 Problem statement and the GILO solution method

The problem of minimizing the Gibbs free energy can be expressed in the primal form shown below. The problem is expressed in a reduced space which is obtained by eliminating one of the substances, say n_s, from each phase, using the fact that the proportions of substances in each phase add up to 1. The following notation is used.

n_s – number of substances; this is fixed.
n – $n_s - 1$.
K – number of phases; this has to be determined.

y^k	– proportion of the total amount which is in phase k.
x_i^k	– proportion of substance i in total amount of phase k.
X	– $\{(v_1,...,v_n) : \sum_{i=1}^n v_i \leq 1,\ v_i \geq 0,\ v_i \in I\!R,\ \ \forall i = 1,...,n\}$, *i.e.* the set of all vectors of proportions of substances.
x^k	– vector of proportions for phase k, *i.e.* $= (x_1^k,...,x_n^k) \in X$.
b_i	– proportion of substance i in the feed.
b	– vector of proportions for the feed. $b \in X$
$F(b)$	– Total equilibrium GFE of system with unit amount of feed with composition b.
$f(x)$	– GFE of unit amount of a phase with composition x.

The amounts of the substances can be measured in units of moles. Given a feed with composition b, the number of phases, K, the total amount in each phase, y^k, and the vector of proportions, x^k, have to be found so as to minimize $F(b)$ globally where

Primal:

$$\begin{aligned} F(b) = \min_{K,y^k,x^k} \quad & \sum_{k=1}^K y^k f(x^k), \\ \text{subject to} \quad & \sum_{k=1}^K y^k x^k = b, \\ & \sum_{k=1}^K y^k = 1, \quad y^k \geq 0, \quad \forall k = 1,...,K, \\ & x^k \in X, \quad \forall k = 1,...,K. \end{aligned} \tag{1}$$

Unfortunately solving this problem as a single global optimization problem is computationally very difficult. The number of phases is not known a priori and can be up to n_s. The problem then has $(n_s - 1)n_s$ variables, and is difficult even for examples with $n_s = 3$, *i.e.* only three substances present. An alternative approach called GILO, global improvement local optimization, is described in [7, 6]. The proofs of convergence in [7] use results in [3] which derive the global optimality conditions for the problem. A related method used in the context of local optimization approaches is described in [8, 9]. The GILO method iterates between a local minimization problem in $K(n_s - 1)$ variables, where $K \leq n_s$, and a global search and verification problem in $n_s - 1$ variables. The global step though still difficult is very much easier than the direct global minimization of the GFE. Under mild assumptions the GILO method is guaranteed to find the global minimum. The main features of the method are described in this paper. More details can be found in [7, 6].

Note that the y^k are non-negative and sum to 1. The point $(F(b), b)$ is therefore a convex combination of the K points $(f(x^k), x^k)$ for $k = 1, ..., K$. Since K is not fixed, $(F(b), b)$ can be any point on the lower boundary of the convex hull of the set $\{(f(x), x) : x \in X\}$. (Note that this set has dimension n_s so no more than n_s points are needed in the convex combination which yields $(F(b), b)$. This shows that the minimum GFE can be achieved with no more that n_s phases.) $F(b)$ is the lower surface of this convex set and so is a convex function. The Lagrangian dual of the primal problem is (see chapter 3 of [12])

Dual:

$$\max_{\lambda \in I\!R^n} V(\lambda), \qquad \text{where}$$

$$V(\lambda) = \min_{x \in X} f(x) - \lambda^T(x - b),$$

where λ is the (column) vector of Lagrange multipliers for the substance balance constraints (1). The function $f(x) - \lambda^T(x-b)$ is the Lagrangian function and will be denoted by $f_\lambda(x)$, *i.e.*

$$f_\lambda(x) = f(x) - \lambda^T(x - b).$$

Note that if f is non-convex, then the inner minimization in the dual is a non-convex problem and there is therefore the possibility that local techniques will not find the global minimum. Also, the outer maximization is non-smooth. Because of this the dual problem is also computationally very difficult.

Let λ^* denote the maximizing value for λ in the dual and let $\overline{F}$ denote the value of the objective function of the primal problem at some feasible point. $\overline{F}$ is therefore an upper bound on $F(b)$. Since $F(b)$ is convex there is no duality gap between the optimal solution $F(b)$ and the solution of the dual, $V(\lambda^*)$, so

$$V(\lambda) \leq V(\lambda^*) = F(b) \leq \overline{F}.$$

It is shown in [3] that the local minima of the Primal occur at Kuhn-Tucker (KT) points. It is shown in [3, 7] that each x^k corresponding to a KT point for which $y_k > 0$ lies in the interior of X, that f is differentiable at such a point and that the Lagrange multipliers have the form $\lambda = \nabla f(x^k)$, and so are independent of k and uniquely determined. It follows that if $\overline{F}$ corresponds to a KT point and λ is the vector of corresponding Lagrange multipliers, then $F(b) = \overline{F} \iff V(\lambda) = \overline{F}$. This can be used to verify that a global minimum has been found. Also if a point x^0 is found such that $f_\lambda(x^0) < \overline{F}$, then since $V(\lambda) \leq f_\lambda(x^0)$, it follows that $\overline{F}$ is not globally optimal. It can also be shown that x^0 can be used to construct a feasible solution to the primal problem which is strictly better than $\overline{F}$ [3, 7, 6].

This is the motivation for the GILO algorithm described below. The GILO algorithm iterates between a (possibly incomplete) global minimization of the inner dual problem (the Global Improvement GI step) and a local optimization of the primal (the Local Optimization LO step). In the GILO algorithm described below, GI is a procedure which will find a point x^0 such that $f_\lambda(x^0) < \overline{F}$ (*i.e* which violates the phase stability criterion), if such a point exists, or will prove that no such point exists and return a point x^0 at which $f_\lambda(x^0) \geq \overline{F}$. The procedure LO is a local minimization procedure for the primal problem which will terminate at a KT point with an objective strictly better than the point from which it was started. (Since the local optimization step is not being parallelized, the full description of the GILO method given in [7, 6] has been shortened here by combining some steps into the LO step.) Provided there are a finite number of distinct function values at KT points and provided GI and LO have the properties described above, this algorithm will converge to the optimum (see [7, 6]).

The GILO algorithm

$y^1 := 1;\ x^1 := b;\ K := 1;\ \overline{F} := f(b)$
do
  $\lambda := \nabla f(x^1)$
  $x^0 := \mathrm{GI}(f_\lambda, \overline{F});\ \ y^0 := 0$
  If $f(x^0) \geq \overline{F}$ then exit
  $(\overline{F}, K, y^1, x^1, ..., y^K, x^K) := \mathrm{LO}(K+1, y^1, x^1, ..., y^K, x^K, y^0, x^0)$
end do

The GI problem can be solved by a variant of a global optimization algorithm using branch and bound. The function f_λ is minimized with an *aspiration level* of $\overline{F}$. Whenever a point x^0 is found with function value below this level, the search stops and returns the point x^0. Any region in which f_λ can be proved to exceed the aspiration level throughout can be eliminated. If a region cannot be eliminated, then it is divided into two, and these subregions are added to a waiting list of regions. The function can be evaluated at a feasible point in each region and these values tested against the aspiration level $\overline{F}$. In this study the bounds are obtained by interval analysis. The feasible region X lies in the set $[0,1]^n$. This set is the initial *box* and this is successively subdivided into smaller non-overlapping boxes. Boxes which do not intersect X are discarded. Bounds on gradients and function values are found using first and second order interval Taylor expansions. Boxes are also eliminated or shrunk using the interval Newton method. The interval arithmetic is implemented in Fortran90 using operator overloading.

In a general branch and bound algorithm for function minimization, it is important to incorporate some aspects of best first search so as to avoid evaluating boxes which could later be eliminated. This is not the case for the GI problem. Any box on the waiting list is there because it has not been shown that all its points exceed the aspiration level. Since the aspiration level never changes, the box will have to be examined further unless the problem terminates with a violation of the phase stability criteria, in which case the GI step immediately terminates and all the waiting list boxes are discarded. It is therefore possible to use an efficient queue or stack data structure to implement the waiting list without this leading to extra boxes being evaluated. Use of a stack gives depth first search, which tends to concentrate the search initially in a small area of X. The danger with this strategy is that the area initially searched may have no low values of f_λ, which would delay the discovery of a point below the aspiration level. We therefore choose to use a (circular) queue for the waiting list. This gives a breadth first search, which samples X in a more uniform manner.

3 Parallelization

For large problems the global improvement (GI) step of the GILO algorithm totally dominates the solution time, so this is the part which has been parallelized. The parallelization uses similar principles to that used for branch and bound in [1]. A related method was used in [10] to parallelize a classical method for global optimization, the multilevel single linkage method.

In a brief initial step, all p processes subdivide the initial box using the normal subdivision rules until each has a single unique box. From then on one process acts as a *central mediator*. It is responsible only for the dynamic load balancing and does no further analysis of boxes itself. The remaining processes, the *workers*, process the boxes using interval analysis and manage their own sections of the search tree, storing all new boxes locally on the worker in their own waiting list L. When a process runs out of work it requests more work from the central mediator. The central mediator keeps a list of boxes to allow it to respond rapidly to such a request, and when this list becomes small it sends a request to the workers for boxes. Whenever the GI step completes without finding a point below the aspiration level, the parallel and serial methods analyze the same boxes and perform the same amount of work.

We will now present the parallel method in more detail. There are three parameters used in the statement of the parallel algorithm: the parameter $\bar{T}$ is the maximum value assigned to T, where T is the minimum length of waiting list on a worker for which that worker responds to a request to return boxes to the central mediator; the parameter $\bar{R}$ is the upper limit to the number of boxes which can be sent in a single message from a worker to the central mediator; and the parameter $\bar{s}$ denotes the upper limit to the number of boxes which can be sent in a single message by the central mediator to a worker. If these parameters are too small, then many short messages will be sent, and if they are too long, then many boxes will be transferred repeatedly between the central mediator and the workers. The algorithm is valid for any values of $\bar{T} \geq 2$, $\bar{R} \geq 1$ and $\bar{s} \geq 1$. The values used in this study are $\bar{T} = 500$, $\bar{R} = 500$ and $\bar{s} = 50$. We have found empirically that, on a network of workstations, these values work well for a wide range of problems. In the following, let p be the number of processes used. We use the notation PBA to denote Point Below Aspiration level.

Parallel algorithm for the worker

```
initial step {Generates subboxes and places a unique one in L}
do
    while L ≠ {}
        if PBA message is waiting then exit procedure
        take first box from L and process it
        if box yields a PBA then
            broadcast PBA message and exit procedure
        if request for boxes from CM is waiting then
            receive this message with its threshold T
            if |L| ≥ T then
                send min(⌊|L|/2⌋, R̄) boxes from list L to CM
            else
                send 'less than T on list' message to CM
    end while
    send request for boxes to CM
    perform a blocking receive for a box-list message from CM
    if received list of boxes is empty then exit
    store received boxes at end of L
    discard any request for boxes that is waiting
end do
```

In the inner loop, boxes on the worker's own waiting list L are processed. The first box is taken out of L, evaluated and, if not eliminated, is subdivided using the same technique as in the serial method. Following this, in order to achieve dynamic load balancing, the worker checks whether a request message for boxes from the central mediator has been received. This message is sent with a threshold T. If the worker has at least T boxes on its waiting list, then the worker sends half of its boxes up to a limit of $\bar{R}$ to the central mediator. The value of T is set by the central mediator to either $\bar{T}$ or 2. The boxes are sent by a nonblocking routine so that the worker does not have to wait until the central mediator has received the message. Once on each pass through the inner loop, a check is made to see if a message has arrived from some other worker indicating that a point has been found violating the aspiration level and if it has the procedure terminates. If a worker's list becomes empty, it leaves the inner loop and sends a request for boxes to the central mediator and then waits to receive boxes. The received boxes are stored in the list L and the worker starts again to consider the boxes of L. If the central mediator sent a message containing no boxes, then this is the signal to terminate the algorithm.

Parallel algorithm for the central mediator

initialize queue $Q := \{\}$ and set counter $r := 0$
initial step $\{$*Generates subboxes and places a unique one in* $L\}$
do
    perform a blocking probe for any message from any worker
    if PBA message waiting then exit
    if 'less than T on list' message is waiting then receive it
    collect requests for boxes and add requesting workers to Q
    collect boxes sent by workers
    if boxes received then $T := \bar{T}$ and $r := 0$
    $s := \max(1, \min(\lfloor \frac{|L|}{(|Q|+1)} \rfloor, \bar{s}))$
    while $|L| > 0$ and $|Q| > 0$
        send $\min(|L|, s)$ boxes to worker on Q and remove from Q
    end while
    if $|L| = 0$ and $|Q| = p - 1$ then exit
    if $|L| \leq \frac{p}{2}$ then
        $r := r + 1$
        if $r > 3(p - 1 - |Q|)$ or all workers not on Q have replied that they
            have fewer than $\bar{T}$ boxes, then $T := 2$
        send request with threshold T for boxes to workers which are not on Q
            except those which have not replied to a previous request
end do
send signal of termination (message with no boxes) to all processors

On the central mediator, a queue, Q, is used to store the worker number of idle workers, and an array is used to record what messages to workers have not yet been answered. The queue Q is initialized to be empty and then a loop is entered. A blocking probe is used to test for any messages having been sent from any of the workers. If there are no messages then the operating system will suspend the process until messages are available. When a message is received the central mediator tests for its type and exits the loop if it is a PBA message. Otherwise the central mediator collects requests for boxes from idle workers and stores the worker's number in the queue Q, collects boxes sent from the workers and stores them at the end of its waiting list L, and collects messages from workers which have not sent boxes as they have less than the threshold T messages on their waiting list. While there are boxes in L and there are outstanding requests for boxes from workers (*i.e.* $|Q| > 0$), a message containing $\max(1, \min(\lfloor \frac{|L|}{(|Q|+1)} \rfloor, \bar{s}))$ boxes taken from the front of L is sent to a worker on Q using a non blocking send and this worker is removed from Q. A check is then made to see if $|L| = 0$ and $|Q| = p - 1$, which only occurs if the waiting lists are all empty, and if they are then the loop is exited. If $\frac{p}{2}$ or fewer boxes are left in the list L then a request is sent using a non blocking send to all processes not on Q (*i.e.* which are not known to be idle) and which have already answered the last message of this kind. Requests for boxes are sent with a threshold level T. This is set to $\bar{T}$ whenever the central mediator receives boxes and is changed to 2 if no boxes have arrived after $3(p - 1 - |Q|)$ (*i.e.* 3 times the number of non idle workers) messages have been received from workers stating that they do not have enough boxes, or once all workers not on Q have replied. This mechanism usually prevents boxes from workers with a short waiting list being sent if other workers have a long list. However it will not block execution if some workers do not respond. When the loop is exited a signal of termination, a message with no boxes, is sent to all workers.

Some aspects of the implementation of the parallel GI algorithm differ from a general parallel method for global minimization such as that described in [1]. As noted in Section 2, L is implemented as a queue. This makes the transfer of boxes between the waiting lists of different processes more efficient than if a partially ordered list were needed. For some problems the total waiting list is over 100000, so it is an advantage to be able to merge and divide the waiting lists in a time that depends linearly on the number of boxes transferred and is independent of the total number on the waiting lists. The waiting list is the only large data structure required for the method. Because it is a queue, memory references to it are local and hence the program's paging behaviour is good, reducing its impact on other users.

Every box inserted into the list L has to be further considered unless the problem terminates. Therefore the length of the list L can only change slowly except during transfers with the central mediator. This is quite different from the case of general global minimization, in which an aspiration level is not known in advance. In that case the length of the list can decrease dramatically when a better upper bound for the global minimum is found. In a cluster of workstations there is a high message latency so there is an advantage in being able to send long messages. For this reason, the message lengths described above are longer than that used in [1] and the waiting list lengths are allowed to become smaller before box transfers occur. The fact that there are no unexpected sudden changes in waiting list lengths makes this long message strategy work well for the GI algorithm. It is also less important in the GI algorithm than in the general global minimization problem to transfer good boxes between processors because all boxes will need some further processing. For this reason the strategy of sending boxes from the front of the list L works well.

The communication protocol used for the central mediator to obtain boxes from the workers does not rely in getting a response from all workers. It copes well with the situation of a worker being temporarily unavailable, which can happen for example if a process has been swapped out of memory to make space for some memory intensive process. The non responding worker cannot hold up the transfer of boxes to idle processes unless it is the only one with boxes.

The central mediator process has been implemented using a blocking probe that waits until any message from any worker process has been received before continuing. When no messages are waiting the operating system suspends the process, so it does not consume resources. In the tests reported in this paper the central mediator and one worker process were placed on one processor. In all cases this produced better results than using the same number of processors but having a single processor dedicated to the central mediator.

4 Results

Results from runs using three difficult test examples are described in this paper. ExA is a 4 substance example with a feed consisting of Ethylene Glycol, Dodecanol, Nitromethane and Water. In equilibrium, there are 4 phases. ExB has 5 substances, the same as in ExA with an additional substance Benzene. ExC has 6 substances, the same as in ExB with an additional substance n-Butanol. ExB and ExC have 3 phases at equilibrium. Even the four substance example is a difficult problem for verified global optimization; it is the most difficult problem considered in [5]. Examples with fewer than 4 substances can be solved so quickly that it is not worthwhile parallelizing their solution.

Table 1 gives the results of running the problems in serial on a 70MHz SUN Sparc5. (All times in the paper are in seconds.) Note that the times are dominated by the GI step and that the major time is in the final GI step, which is the only one which does not terminate early on finding a point violating the aspiration level.

The parallel implementation uses the MPI message passing protocol to link a cluster of workstations on an ethernet. Four types of SUN workstation were used: Sparc5 (SS5),

	ExA		ExB		ExC	
	No of iters	CPU time	No of iters	CPU time	No of iters	CPU time
1^{st} Global Step†	6	0.03	8	0.06	8	0.08
1^{st} Local Step	8	0.04	6	0.05	5	0.05
2^{nd} Global Step	15	0.08	856	7.18	5888	87.20
2^{nd} Local Step	129	0.82	159	1.78	17	0.32
3^{rd} Global Step	3016	15.65	57450	667.36	1069179	19983.08
3^{rd} Local Step	5	0.07				
4^{th} Global Step	7975	54.22				
Entire Program		71.05		677.37		20071.41

† : vapour phase introduced.

Table 1: Iteration counts and serial times (sec) on a Sparc5.

ELC, Sparc4 (SS4) and Quad Hypersparc (QHS). A QHS workstation has four hypersparc processors and shared memory. Its operating system allocates processes to processors dynamically and a single process will be processed on different processors at different times. All processors of the same type have the same performance. Since the workstations have different powers and may be being used by other users we need to develop a calibration procedure to be able to assess the performance of the parallel implementation.

The relative power P_p of each processor was assessed by measuring the CPU time to solve ExB in serial on a single processor of each type and calculating $P_p =$ (CPU time on processor p)/(CPU time on a SS5). (Note that on the QHS this is the time using at any one time only one of the 4 processors.) This gives the values $P_{SS5} = 1$, $P_{ELC} = 0.319$, $P_{SS4} = 1.360$ and $P_{QHS} = 2.250$.

Because other jobs may be running simultaneously on workstations, we need to estimate processor *availability*, *i.e.* the proportion of a processor which is available to a process when it has work to do. To find the availability and to obtain performance measures for the parallel algorithm, we measure and calculate the following quantities over the final GI step:

T_{ser}^{CPU} – the serial CPU time.

T_{par}^{ELP} – the elapsed time for the parallel run (*i.e.* the wall clock interval).

T_p^{CPU} – the total CPU time on processor p.

T_{par}^{CPU} $= \sum_p P_p T_p^{CPU}$, *i.e.* the total normalized CPU time for the parallel run.

t_p^{CPU} – the sum of the CPU times for each worker process p over all intervals when the process was processing boxes, and $= T_p^{CPU}$ for the central mediator.

t_p^{ELP} – the sum of the elapsed times for each worker process p over all intervals when the process was processing boxes, and $= T_{par}^{ELP}$ for the central mediator.

a_p $= t_p^{CPU}/t_p^{ELP}$, *i.e.* availability.

P $= \sum_p a_p P_p$, *i.e.* total machine power available for the run. P can be interpreted as the number of otherwise unused SS5s needed to achieve the same processing rate as the combination of loaded workstations used for the run.

S $= T_{ser}^{CPU}/T_{par}^{ELP}$, *i.e.* the speedup.

U $= S/P$, *i.e.* the utilization.

E $= T_{ser}^{CPU}/T_{par}^{CPU}$, *i.e.* the efficiency.

There is a small fixed amount of work for the central mediator, and when no work is available the process is suspended and uses no resources. The central mediator therefore takes a small fixed amount of resource on the processor on which it runs. It follows that the appropriate measure of availability for the central mediator process is obtained by using the elapsed time for the entire run, rather than just those intervals when work for the central mediator is available. The maximum possible value for U is 1.0, which could occur only if there were no delays due to communication and no extra work done – all available machine cycles used for processing the boxes. Utilization becomes the same as the concept of parallel efficiency in the single user case. The quantity $(1 - E)$ gives, as a proportion, the extra CPU effort needed by the parallel algorithm to process messages and box transfers. Incomplete load balancing leading to idle processes causes U to be less than E. (Note that CPU timings on SUN workstations are not highly accurate and can vary by several percent between identical runs. Also the clock calls, which are not part of the algorithm but only there to obtain the performance estimates, degrade the parallel performance of ExA, ExB and ExC by 3.5%, 2.0% and 1.3%. To adjust for this, the serial times given in Tables 2 and 3 have been increased by these percentages over that actual times given in Table 1.)

For the sake of other users of the network it is important that the algorithm does not make excessive demands on the ethernet. To assess the communication load of the algorithm, we measure the total number of messages sent, m, the number of box transfers between processes, b, the total length in bytes of all messages sent, l, and the total number of boxes processed, i, which is also the number of iterations. These are then used to calculate the measures Box-Tran/Iter $= b/i$, Messages/Iter $= m/i$ and Ether-Use% $= (8.3 \times 10^{-2} m + 8.5 \times 10^{-5} l)/T_{par}^{ELP}$. (These ethernet parameters were estimated using the spray utility on an otherwise quiet ethernet.) Note that in all the runs the central mediator and one of the worker processes were placed on the same processor. The communication between these processes therefore did not use the ethernet and so the ethernet use was slightly less than the estimate. However the central mediator was never placed on the QHS, so communication between the central mediator and the four processes on the QHS used the ethernet in all the runs.

The results of one run of ExB, Run 3, are presented in detail and the remaining results in summary form. Run 3 used 5 processors; 3 SS5s, 1 ELC and one of the QHS processors. The central mediator and one worker was placed on a SS5 and the other 4 workers were allocated to different machines. The upper section of Figure 1 shows Gantt charts of the CPU usage and the lower section shows the waiting list lengths, and table 2 shows detailed timings and utilization and communication results for the same run.

The vertical lines on the Gantt charts for the workers show the points at which messages containing boxes are sent or received by that process and gaps within the GI steps show intervals when the process has run out of boxes or is delayed by communicating boxes to or from the central mediator. Gaps on the Gantt chart for the central mediator show intervals during which the central mediator's waiting list is empty. The proportion of the chart that is unshaded gives the proportion of the processor's CPU that is being used by that process during the interval between the adjacent vertical lines. Note that the central mediator starts first and there is very little CPU use in the first period. This is because there is user interaction and the data is being read. After this there is a LO step on the central mediator (shown as a small gap), a global GI step on all processors which terminates early, a further LO step on the central mediator (shown as a gap), and the final GI step on all processors. Note that during the GI steps there are very few idle periods for workers which are long enough to create a visible gap in the charts. Whenever a worker process has boxes to process and is the only process running on a processor, almost all the available CPU time on that processor is taken by the worker process. The sections of the workers' charts where the usage is less that 100% are therefore due to other processes on the same processor. It follows that during Run 3 the workstation cluster was relatively busy with other work. The reason why the central mediator chart shows very low CPU usage is different. This is because the central mediator process was suspended for most of the time waiting for messages from workers and on the central mediator the gaps show only intervals where the waiting list was empty. The bottom half of Figure 1 shows graphs of the waiting list lengths. There are only a small number of points

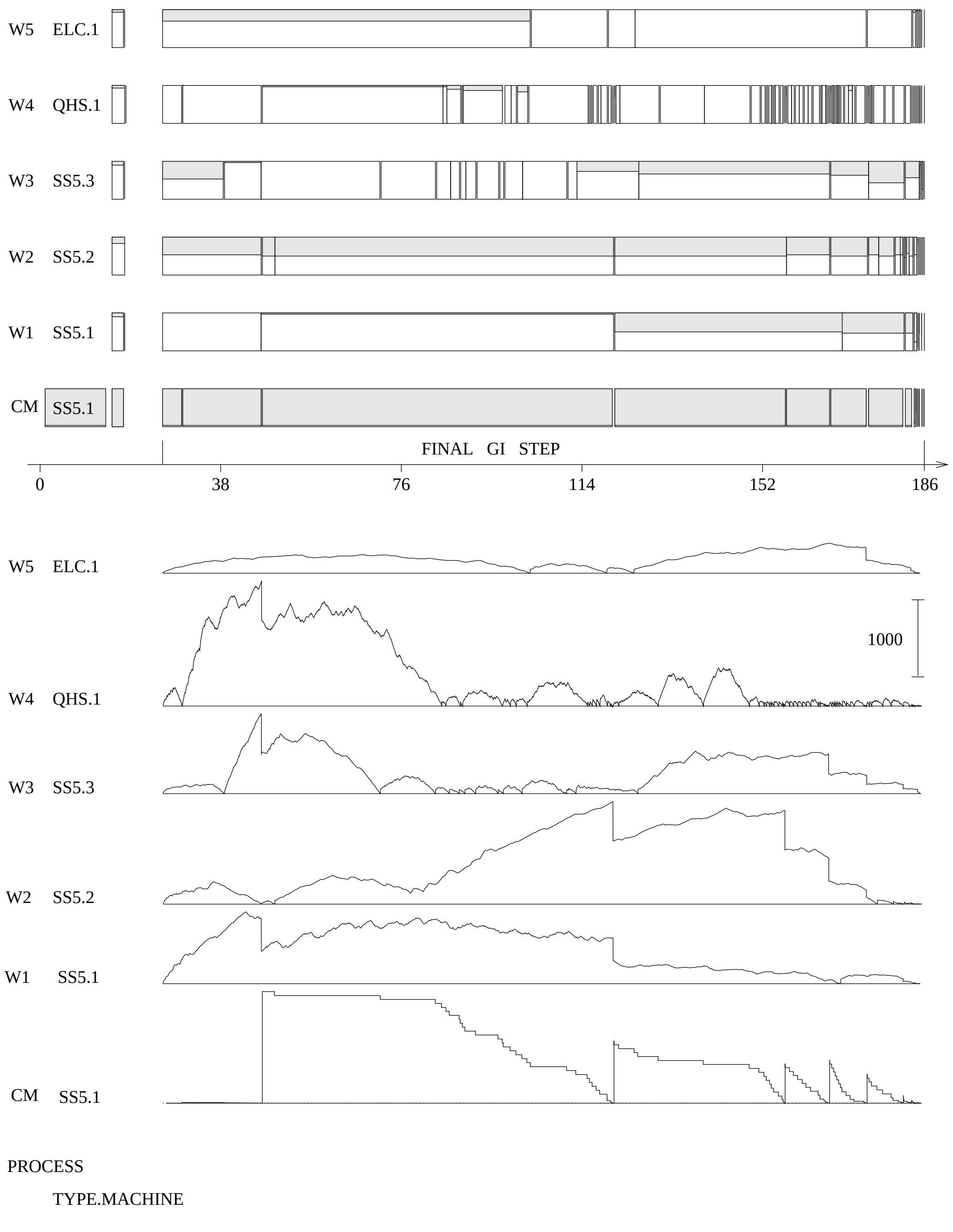

Figure 1: Run 3, ExB. Top – CPU usage; Bottom – Waiting list size. Time(sec).

W/CM (p)		Type.Machine	T_p^{CPU}	t_p^{CPU}	t_p^{ELP}	a_p	P_p
W5		ELC.1	133.94	129.40	155.00	0.835	0.319
W4		QHS.1	152.15	145.09	151.03	0.961	2.250
W3		SS5.3	126.14	118.39	152.90	0.774	1.000
W2		SS5.2	79.05	74.52	151.10	0.493	1.000
W1	‡	SS5.1	120.25	114.62	153.01	0.749	1.000
CM	‡	SS5.1	2.14	2.14	160.82	0.013	1.000

‡ : Same processor.
$T_{ser}^{CPU} = 681.02$, $T_{par}^{ELP} = 160.82$, $T_{par}^{CPU} = 712.64$, Speedup $(S) = 4.23$,
Power $(P) = 4.45$, Utilization $(U) = 0.952$, Efficiency $(E) = 0.956$,
Box Transfers/Iteration = 0.13, Messages/Iteration = 0.0080,
Ethernet Use = 0.61%.

Table 2: Run 3, ExB: Parallel results for final global step.

at which the waiting list lengths change significantly, indicating large numbers of boxes being transferred to the central mediator. Table 2 shows the timings needed to calculate the performance measures. The total power (P) of the workstation cluster in this run is equivalent to 4.45 unloaded SS5s. The speedup (S) achieved is 4.23 and the utilization (U) of this available resource is 0.952. The efficiency (E) is 0.956, which is close to the utilization, indicating that the load balance is good. This is confirmed by the absence of significant gaps in the Gantt charts for the workers in Figure 1. The number of boxes transferred and number of messages are low leading to a very low estimate of 0.6% for ethernet use.

Table 3 shows the performance of the algorithm for the three test problems on different workstation combinations and workstation loadings. Runs 1, 4 and 7 show results for the three examples on 7 Sparc5s. Runs 2, 5 and 8 show the three examples run on 7 Sparc5s, 2 ELCs, 1 Sparc4 and with four processes on the four processor QHS. Run 6 shows the result of running example B on 7 Sparc5s and 7 ELCs. All runs have the central mediator on the same Sparc5 processor as one of the workers.

Speedup results in all cases are good. The smallest example ExA has been solved (in Run 2) with a speedup of 10.4 in 5.4 seconds on 14 processors. This run has the poorest utilization of 0.65. In the other runs this varies from 0.89 to 0.97. Run 2 is also the only case in which the ethernet use is significant (12.5%). In the other runs it is less than 4%. The efficiency and utilization figures are lowest for ExA. The efficiencies are larger and the ethernet use less for ExB and ExC. Separate measurements show that the central mediator takes between 2% and 3% of the total CPU time for ExA but less that 0.5% for the other examples. Testing for receipt of messages take about 4%, 2% and 1% of the CPU time for ExA, ExB and ExC respectively. The utilization and efficiency figures differ significantly only for Runs 1, 2 and 5, indicating that it is only on these runs that there is some load imbalance. The difference is negligible for ExC. The loss in efficiency is due to the extra load on the CPUs caused by transferring boxes and testing for receipt of messages.

The results given in Table 3 show that the the number of boxes transferred and the number of messages per iteration are low. An algorithm in which a master process held

Run	1	2	3	4
Ex	A	A	B	B
Machines	1 $SS5^{CM}_{1W}$ 6 $SS5_{1W}$	1 $SS5^{CM}_{1W}$ 6 $SS5_{1W}$ 1 $SS4_{1W}$ 2 ELC_{1W} 1 QHS_{4W}	1 $SS5^{CM}_{1W}$ 2 $SS5_{1W}$ 1 ELC_{1W} 1 QHS_{1W}	1 $SS5^{CM}_{1W}$ 6 $SS5_{1W}$
Serial Time	56.12	56.12	681.02	681.02
Elapsed Time	11.17	5.39	160.82	118.29
Speedup	5.024	10.412	4.235	5.757
Power	5.667	15.910	4.447	6.106
Utilization	0.887	0.654	0.952	0.943
Efficiency	0.906	0.839	0.956	0.944
Box-Tran/Iter	0.197	0.247	0.133	0.132
Messages/Iter	0.0498	0.0921	0.00795	0.00888
Ether-Use (%)	3.94	12.54	0.608	0.860
Run	5	6	7	8
Ex	B	B	C	C
Machines	1 $SS5^{CM}_{1W}$ 6 $SS5_{1W}$ 1 $SS4_{1W}$ 2 ELC_{1W} 1 QHS_{4W}	1 $SS5^{CM}_{1W}$ 6 $SS5_{1W}$ 7 ELC_{1W}	1 $SS5^{CM}_{1W}$ 6 $SS5_{1W}$	1 $SS5^{CM}_{1W}$ 6 $SS5_{1W}$ 1 $SS4_{1W}$ 2 ELC_{1W} 1 QHS_{4W}
Serial Time	681.02	681.02	20237.03	20237.03
Elapsed Time	45.57	82.14	3603.96	1243.04
Speedup	14.944	8.291	5.615	16.280
Power	16.189	8.886	5.938	16.726
Utilization	0.923	0.935	0.946	0.973
Efficiency	0.955	0.949	0.950	0.975
Box-Tran/Iter	0.170	0.175	0.174	0.188
Messages/Iter	0.0181	0.0254	0.00430	0.00578
Ether-Use (%)	3.58	2.44	0.579	1.90

Key: $m\ M^{\{CM\}}_{rW}$– m machines of type M with r worker processes {and central mediator}

Table 3: Parallel results for final GI step.

the entire waiting list and sent out a single box at each iteration, would have box transfer and message per iteration rates about one and two orders of magnitude higher than the algorithm used in this study, which is based on a central mediator.

As noted earlier, in the final GI step the serial and parallel algorithms evaluate exactly the same boxes. This is not however the case for the GI steps which terminate early having found a point below the aspiration level. For these GI steps the course of the search depends on what type of machine gets which part of the initial search space and how busy it is with other jobs. For the examples used in this paper, the number of iterations taken in the parallel case has been observed to vary between 0.5 and 4 times the number taken in the serial case. There is no systematic pattern to the number of iterations taken, which can vary widely between different runs of the same example. The speedup varies between 2 and 10, with the higher speedups occurring in those examples where the GI step takes a long time in the serial case, and in these cases the speedups are similar to those for the final GI step. Therefore the GI steps which terminate early do not have a large affect on the speedup of the entire algorithm, which is in all cases close to the value for the final GI step.

5 Conclusions

A parallelization of the Global Improvement Local Optimum (GILO) algorithm for minimizing the Gibbs free energy of a mixture has been presented. Problems with mixtures of up to 6 substances can be solved to proven optimality in reasonable times on a cluster of workstations. The GILO algorithm iterates between a primal local minimization and a dual Global Improvement (GI) step. The GI step uses interval arithmetic to obtain bounds. The GI step dominates the solution time and is the part which has been parallelized. The approach to parallelization uses a central mediator for achieving good load balancing. The GI problem differs from a general global optimization problem in that an aspiration value for the objective is known a priori and if this is violated the search can be terminated immediately. This has been exploited to create a more efficient parallel algorithm. The paper shows that the algorithm that has been developed is robust and well suited for use on a cluster of workstations. It is able to cope efficiently with processes which are temporarily suspended as well as with different machine speeds and changing workloads on each machine due to other users. To assess the performance of the algorithm a method has been defined to determine the utilization of a parallel method on a heterogeneous cluster of workstations which may be in use by other users. This measures the proportion of the available resource which was used for the core computation. Measurements show utilizations of 89% and 65% for the small problem, 94% and 92% for the medium sized problem and 95% and 97% for the large problem, using 7 and 14 processors respectively.

The network traffic generated by the method is very low, so the method has a low impact on other network users. The use of the central mediator reduces the communication by a factor of between 10 and 100 compared with the alternative of storing the entire waiting list on the master and sending a single box to a worker in each iteration.

6 Acknowledgements

This work was supported by the UK EPSRC grant RR31817/18 (the ECOSSE programme) and the EC Human Capital Mobility grant ERB-CHCE-CT92-005 (the EPCC's TRACS programme). The authors are grateful to Julian Hall for providing his software for producing Gantt charts, which has been invaluable for understanding the algorithm, and to the referees for helpful and detailed comments, which improved the paper.

References

[1] S. Berner. Parallel methods for verified global optimization — practice and theory. *Journal of Global Optimization*, 9:1–22, 1996.

[2] A. Fredenslund, J. Gmehling, and P. Rasmussen. *Vapor-liquid equilibria using UNIFAC a group-contribution method.* Amsterdam Oxford Elsevier, 1977.

[3] Y. Jiang, W. Smith, and G. Chapman. Global optimality conditions and their geometric interpretation for the chemical and phase equilibrium problem. *SIAM Journal on Optimization*, 5(4):813–834, 1995.

[4] T. Magnussen, P. Rasmussen, and A. Fredunslund. UNIFAC parameter table for prediction of liquid-liquid equilibria. *Ind. Eng. Chem. Process Des. Dev.*, 20:331–339, 1981.

[5] C. McDonald and C. Floudas. GLOPEQ: A new computational tool for the phase and chemical equilibrium problem. *Computers & Chemical Engineering*, 21(1):1–23, 1996.

[6] K. McKinnon, C. Millar, and M. Mongeau. Global optimization for the chemical and phase equilibrium problem using interval analysis. In C. Floudas and P. Pardalos, editors, *State of the Art in Global Optimization*, pages 365–382. Kluwer Academic Press, 1996.

[7] K. McKinnon and M. Mongeau. A generic global optimization algorithm for the chemical and phase equilibrium problem. Technical Report MS 96-011, Dept. of Mathematics and Statistics, University of Edinburgh, U.K., 1996. http://www.maths.ed.ac.uk/mckinnon/.

[8] M. L. Michelsen. The isothermal flash problem. Part I. Stability. *Fluid Phase Equilibria*, 9:1–19, 1982.

[9] M. L. Michelsen. The isothermal flash problem. Part II. Phase-split calculation. *Fluid Phase Equilibria*, 9:21–40, 1982.

[10] S. Smith and R. Schnabel. Dynamic scheduling strategies for an adaptive, asynchronous parallel global optimization algorithm. Technical Report CU-CS-625-92, University of Colorado, 1992.

[11] W. Smith and R. Missen. *Chemical Reaction Equilibrium Analysis: Theory and Algorithms.* Wiley & Sons, 1982.

[12] P. Whittle. *Optimization Under Constraints.* Wiley-Interscience, 1971.

7 Appendix: UNIFAC model, problem data and optimal solutions

7.1 UNIFAC model

The results in this paper were obtained using the UNIFAC model [2] for the GFE of a phase. According to this model the Gibbs free energy for a single phase has the form

$$
\begin{aligned}
f(x) &= \min\{f^L(x), f^V(x)\}, \quad \text{where,} \\
f^L(x) &= \sum_{i=1}^{n} x_i \left\{ \ln x_i + \ln P_i^{sat} - \mu_i^R r_i \ln r_i - v^{(i)} + \mu_i^R r_i \ln(\sum_{j=1}^{n} r_j x_j) \right. \\
&\qquad \left. + q_i(1 - \frac{\mu}{2}) \ln(\sum_{j=1}^{n} q_j x_j) + \frac{\mu}{2} q_i \ln q_i - \sum_{l=1}^{n_G} v_{li} Q_l \ln(\sum_{j=1}^{n} \hat{v_{lj}} x_j) \right\}, \\
f^V(x) &= \sum_{i=1}^{n} x_i \{\ln x_i + \ln P\}.
\end{aligned}
$$

n_G is the number of groups, P_i^{sat} is the saturated vapour pressure of component i, P is the pressure and μ is the lattice coordination number. The phase-class model for liquid is f^L and for vapour is f^V. The constants are obtained from the tables in section 7.2 and from the following relations

$$q_i = \sum_{l=1}^{n_G} v_{li} Q_l, \quad r_i = \sum_{l=1}^{n_G} v_{li} R_l, \quad \mu_i^R = \frac{\frac{\mu}{2} q_i - 1}{r_i}, \quad \mu = 10.$$

7.2 Data for UNIFAC model

The data for the test examples is given in Tables 4 and 5. The values are calculated from values given in [4]. All examples correspond to a temperature of 350K, and a pressure P of 0.43atm.

7.3 Data for examples and optimal solutions

The optimal solution to the test examples and the data specific to them is given in Tables 6, 7 and 8.

l		R_l	Q_l
1	$(CH_2OH)_2$	2.4088	2.248
2	CH_2	0.6744	0.54
3	CH_3	0.9011	0.848
4	OH	1.0	1.2
5	CH_3NO_2	2.0086	1.868
6	H_2O	0.92	1.4
7	ACH	0.5313	0.4

Table 4: R and Q parameters for UNIFAC model

i	$\hat{v}_{1i}$ (v_{1i})	$\hat{v}_{2i}$ (v_{2i})	$\hat{v}_{3i}$ (v_{3i})
1	2.24800000000000 (1)	1.64878875759220 (0)	1.64878875759220 (0)
2	1.76285488194877 (0)	7.25782606421081 (11)	7.25782606421081 (1)
3	1.25287356923659 (0)	1.42664492691213 (0)	1.42664492691213 (0)
4	3.02108214621398 (0)	0.52633705753912 (0)	0.52633705753912 (0)
5	0.86074254825167 (0)	1.53468612332395 (0)	1.53468612332395 (0)
6	0.80579321792348 (0)	2.93782606421081 (3)	2.93782606421081 (1)
i	$\hat{v}_{4i}$ (v_{4i})	$\hat{v}_{5i}$ (v_{5i})	$\hat{v}_{6i}$ (v_{6i})
1	5.13773520414380 (0)	0.56830221953843 (0)	7.55588436898295 (0)
2	2.27620618126340 (1)	2.69346768407895 (0)	1.27087446066458 (0)
3	0.46887579526004 (0)	1.86800000000000 (1)	1.79824501379086 (0)
4	1.98612702209297 (0)	0.01723745544700 (0)	1.40000000000000 (1)
5	0.32120551733851 (0)	2.41216788428922 (0)	0.20598337155145 (0)
6	1.59129004940455 (1)	1.81301230252324 (0)	1.16558377426801 (0)
i	$\hat{v}_{7i}$ (v_{7i})	$v^{(i)}$	P_i^{SAT}
1	1.10901197182141 (0)	0	$6.09569441739 \times 10^{-3}$
2	10.6550230200592 (0)	2.157214543937	$2.81223352086 \times 10^{-4}$
3	1.35025965532709 (0)	0	0.44338413373
4	0.48255005602760 (0)	0	0.41099747914
5	2.40000000000000 (6)	0	0.90370484174
6	4.65804665976889 (0)	1.549960265842	0.18416553048

Table 5: Table of v, $\hat{v}$ and P^{SAT} values for all examples

Ethylene Glycol(1), Dodecanol(2), Nitromethane(3), Water(4)

Component	i	z_i	Liquid I (mols)	Vapour (mols)
$(CH_2OH)_2$	1	0.3	0.2229026	1.852531×10^{-3}
$C_{12}H_{25}OH$	2	0.1	2.148462×10^{-3}	7.296264×10^{-5}
CH_3N0_2	3	0.5	9.787671×10^{-3}	0.2109185
H_2O	4	0.1	8.097318×10^{-2}	8.550184×10^{-3}
Component	i	z_i	Liquid II (mols)	Liquid III (mols)
$(CH_2OH)_2$	1	0.3	5.583693×10^{-2}	1.940795×10^{-2}
$C_{12}H_{25}OH$	2	0.1	9.413635×10^{-2}	3.642225×10^{-3}
CH_3N0_2	3	0.5	7.254400×10^{-2}	0.2067499
H_2O	4	0.1	7.507998×10^{-3}	2.968635×10^{-3}

$n = 4$, $n_G = 6$. Optimal $K = 4$, F = -3.430398

Table 6: Optimal solution for ExA

Ethylene Glycol(1), Dodecanol(2), Nitromethane(3), Water(4), Benzene(5)

Component	i	z_i	Liquid I (mols)	Vapour (mols)	Liquid II (mols)
$(CH_2OH)_2$	1	0.2	4.522995×10^{-2}	1.270658×10^{-3}	0.1534994
$C_{12}H_{25}OH$	2	0.2	0.1993314	1.083779×10^{-4}	5.602098×10^{-4}
CH_3N0_2	3	0.2	5.084352×10^{-2}	0.1473621	1.794376×10^{-3}
H_2O	4	0.2	1.728813×10^{-2}	4.207621×10^{-2}	0.1406357
C_6H_6	5	0.2	7.547863×10^{-2}	0.1238629	6.584384×10^{-4}

$n = 5$, $n_G = 7$. Optimal $K = 3$, F = -4.281736

Table 7: Optimal solution for ExB

Ethylene Glycol(1), Dodecanol(2), Nitromethane(3), Water(4), Benzene(5), n-Butanol(6)

Component	i	z_i	Liquid I (mols)	Vapour (mols)	Liquid II (mols)
$(CH_2OH)_2$	1	0.18	6.147480×10^{-2}	1.063684×10^{-3}	0.1174615
$C_{12}H_{25}OH$	2	0.18	0.1792704	7.831926×10^{-5}	6.512598×10^{-4}
CH_3N0_2	3	0.18	5.431986×10^{-2}	0.1237493	1.930751×10^{-3}
H_2O	4	0.18	2.883324×10^{-2}	4.079167×10^{-2}	0.1103751
C_6H_6	5	0.18	7.241087×10^{-2}	0.1067994	7.897762×10^{-4}
C_4H_9OH	6	0.1	7.092728×10^{-2}	1.990080×10^{-2}	9.171915×10^{-3}

$n = 6$, $n_G = 7$. Optimal $K = 3$, F = -4.294716

Table 8: Optimal solution for ExC