Journal of Universal Computer Science, vol. 7, no. 4 (2001), 291-306
submitted: 14/7/00, accepted: 17/4/01, appeared: 28/4/01 © Springer Pub. Co.

The Message-Minimizing Load Redistribution Problem

David J. Haglin
(Minnesota State University, Mankato, U.S.A.
currently visiting The University of Manchester, U.K.
david.haglin@mnsu.edu)

Rupert W. Ford
(Centre for Novel Computing,
The University of Manchester, U.K.
rupert@cs.man.ac.uk)

Abstract: The Message Minimizing Load Redistribution Problem is described which arises from the need to redistribute work when performing load balancing in a parallel computing environment. We consider a global perspective and seek a redistribution plan that minimizes the overall processing time. We define the cost associated with a solution to be the number of packets needed to balance out the workload. The impact of the interconnection network is ignored. This problem can arise in many applications. One such example being the U.K. Meteorological Office's operational weather forecasting and climate prediction models.

This problem is equivalent to the *Pure Unit-Cost Transportation Problem*. A simple proof of $\mathcal{NP}$-completeness is given, and various heuristics and approximation issues are investigated. Several theoretical results are shown that may impact the design of an algorithm. Simulation results are presented.

Key Words: Parallel Processors, High Performance Computing, Load Balancing
Category: C.1.2, C.2.4, F.2.2, G.2.1, J.2

1 Introduction

The *message minimizing load redistribution problem* (MMLRP) occurs in a parallel or distributed computing environment where load balancing may improve the overall performance, and where data must be transferred from processor to processor as part of the load redistribution. We consider a global perspective and seek an optimum load balance after redistribution. We therefore have a fixed amount of data to move, so we endeavor to minimize the number of communication messages needed to achieve the redistribution.

Given a situation where some processors have more work to do than others, redistributing the workload will decrease the total elapsed time. Those processors with more work than the average workload are called *overloaded*, and the others are called *underloaded*. A *solution* to this problem is a plan for sending work from the overloaded processors to the underloaded processors in such a way as to achieve the best possible load balance. Clearly we would like to find the fastest solution for achieving the required load balance. Noting that all solutions send the same amount of data, our strategy is to find a plan that does this using the fewest communication messages.

The MMLRP can be restated using transportation terminology. Let μ be the ceiling of the average workload among the processors. The overloaded processors are *source* sites with a *supply* equal to the work load in excess of μ. And the underloaded processors are *destination* sites with a *capacity* of μ minus the work load.

The rest of this paper is organized as follows: [Section 2] gives definitions used throughout the rest of the paper, [Section 3] shows theoretical results, [Section 4] discusses several algorithms and heuristics, [Section 5] describes an application, [Section 6] provides experimental results, and [Section 7] concludes the paper.

2 Definitions

An instance of MMLRP is a set of n sources with item quantities (*supply*) given for each source and a set of m destinations with item *capacities* given for each destination. Such an instance will be denoted $\mathcal{I} = (S, D)$, where S are the sources and D are the destinations, which gives rise to a complete bipartite graph $G = (S, D, A)$, where S are the source vertices, D are the destination vertices, and $A = \{(u, v) : u \in S, v \in D\}$. Each vertex $u \in S$ has an associated item supply $w(u)$ and each vertex $v \in D$ has an associated capacity $w(v)$. Where no ambiguity arises, we write the collection $\{w(u) | u \in S\}$ as a set $S = \{s_1, \ldots, s_n\}$ or as a sequence $S = s_1, \ldots, s_n$. The destinations $\{w(v) | v \in D\}$ are similarly denoted as a set $D = \{d_1, \ldots, d_m\}$ or as a sequence $D = d_1, \ldots, d_m$. To indicate the range $1 \leq i \leq n$, we will write $i \in S$. Similarly, for $1 \leq j \leq m$ we will write $j \in D$. To ensure that the total of supply items and capacities are acceptable (i.e. the problem has a feasible solution), we require that $\sum_{i=1}^{n} s_i \leq \sum_{j=1}^{m} d_j$.

A *feasible solution* X is an $n \times m$ integer matrix with the following constraints:

$$\begin{array}{ll} \forall i : \sum_{j=1}^{m} x_{i,j} = s_i & \text{(redistribution)} \\ \forall j : \sum_{i=1}^{n} x_{i,j} \leq d_j & \text{(capacity)} \\ \forall i, j : x_{i,j} \geq 0 & \text{(non-negative flow)} \end{array}$$

The *cost* of a solution, which is called the *objective function* and is denoted by $|X|$, is the number of non-zero entries in the matrix. The problem, then, is to find an X of smallest cost satisfying all of the constraints. We will refer to a non-zero entry in the matrix as a *message* carrying $x_{i,j}$ items.

2.1 The Transportation Problem

The MMLRP is related to the classic transportation problem (TP), which is an important optimization problem in Operations Research (cf. [2]). The TP can be thought of as the task of transporting *items*, all of which are identical, from a collection of *sources* to a collection of *destinations*. Initially, the items are distributed among the various sources (i.e. each source has a given *supply*), and each of the destinations has a given maximum *capacity*. The sum of the destination's capacities must be at least as large as the sum of the supply. The costs for the TP is given as $c_{i,j}$, and the objective function is $\sum c_{i,j} x_{i,j}$.

For situations where there is a fixed cost associated with using a source-to-destination pair (i.e. at least one item is to be moved) as well as a per-unit cost of transportation, the cost function can be given by:

$$cost(i,j) = \begin{cases} h_{i,j} + c_{i,j} \cdot x_{i,j} & \text{if } x_{i,j} > 0 \\ 0 & \text{if } x_{i,j} = 0 \end{cases}$$

where $h_{i,j}$ is the fixed cost of using the link from source i to destination j.

This modified problem is known as the *fixed charge transportation problem* (FCTP). If there is no per-unit cost, that is $c_{i,j} = 0$, then the problem is known as the *pure fixed charge transportation problem* (PFCTP). The MMLRP is a special case of the PFCTP where the fixed cost is always one ($h_{i,j} = 1$). Another application with this same formulation is called the *teacher assignment problem* [7]. We refer to this special case as the *pure unit-cost transportation problem* (PUCTP). We will interchangeably use the terms MMLRP and PUCTP.

2.2 Mathematical Programming Formulation

Solving the PUCTP exactly is not appropriate for most applications, due its computational complexity. In a MMLRP application, the cost of computing a transportation plan must be weighed against the benefit in reduced communication cost. So finding a quick approximation is suitable to our needs. However, we look at formulating this problem in the language of Mathematical Programming.

We begin with the standard transportation problem formulation.

$$(\text{TP}) \qquad \begin{cases} \text{Minimize} & \sum_{i=1}^{n} \sum_{j=1}^{m} c_{i,j} \cdot x_{i,j} \\ \\ \text{subject to} & \sum_{j=1}^{m} x_{i,j} = s_i, \quad i \in S \\ & \sum_{i=1}^{n} x_{i,j} = d_j, \quad j \in D \\ & x_{i,j} \geq 0, \quad i \in S, j \in D \end{cases}$$

where $c_{i,j}$ is the per-item cost of transporting from source i to destination j.

A standard transformation to a FCTP is to add a set of 0-1 variables $y_{i,j}$, where $y_{i,j} = 1$ if and only if $x_{i,j} > 0$. The standard general FCTP has a fixed charge $h_{i,j}$ associated with each link.

$$\text{(FCTP)} \quad \begin{cases} \text{Minimize} & \sum_{i=1}^{n}\sum_{j=1}^{m} h_{i,j}\cdot y_{i,j} + \sum_{i=1}^{n}\sum_{j=1}^{m} c_{i,j}\cdot x_{i,j} \\ \text{subject to} & \sum_{j=1}^{m} x_{i,j} = s_i, \qquad i \in S \\ & \sum_{i=1}^{n} x_{i,j} = d_j, \qquad j \in D \\ & x_{i,j} \le u_{i,j}\cdot y_{i,j}, \quad i \in S, j \in D \\ & y_{i,j} \in \{0,1\}, \qquad i \in S, j \in D \end{cases}$$

where $u_{i,j} = \min(s_i, d_j)$ for all $i \in S, j \in D$ (cf. [11]). Setting $c_{i,j} = 0$ and $h_{i,j} = 1$ for all $i \in S, j \in D$, we have a formulation of our problem.

$$\text{(PUCTP)} \quad \begin{cases} \text{Minimize} & \sum_{i=1}^{n}\sum_{j=1}^{m} y_{i,j} \\ \text{subject to} & \sum_{j=1}^{m} x_{i,j} = s_i, \qquad i \in S \\ & \sum_{i=1}^{n} x_{i,j} = d_j, \qquad j \in D \\ & x_{i,j} \le u_{i,j}\cdot y_{i,j}, \quad i \in S, j \in D \\ & y_{i,j} \in \{0,1\}, \qquad i \in S, j \in D \end{cases}$$

Recall that our work items are integral, so we insist that all $x_{i,j}$ be integral.

2.3 Observations

We may assume, without loss of generality, that $m > n$ for all instances due to the following observation.

Proposition 1. *Every problem has an equivalent corresponding dual problem.*

Note that any solution moving items from the sources to the destinations is also a solution to the problem where the sources and destinations are swapped (with appropriate item capacities and quantities interchanged). Thus, a solution to the dual problem is also a solution to the original problem after inverting the direction of item transport. □

To make the theoretical analysis easier, we may assume that the total supply is exactly equal to the total capacity.

Proposition 2. *If* $\Delta = \sum_{j=1}^{m} d_j - \sum_{i=1}^{n} s_i > 0$, *an equivalent problem can be formulated by adding* Δ *sources, each with one item to transport.*

Note that any solution to the original problem would have left places for Δ items among the destinations. Then, the dummy sources' supply could fill up these gaps completely. This is the same cost contribution no matter what the solution to the original problem. □

A trivial lower bound can be seen by noting that each source must send at least one message, and each destination must receive at least one message.

Proposition 3. *Every solution has at least* $\max(n, m)$ *messages.* □

3 Theoretical Results

3.1 $\mathcal{NP}$-completeness

The PUCTP problem is already known to be $\mathcal{NP}$-complete [7]. But their proof involves a reduction through an alternative problem called the *maximum cardinality 2-dimensional partition* problem. Here we give a simple and direct proof that PUCTP is $\mathcal{NP}$-complete.

Theorem 4. *The PUCTP is $\mathcal{NP}$-complete.*

Proof. We reduce from the INTEGER PARTITION (IP) problem. Given any instance of the IP problem (I_{IP}): $a_1, a_2, \ldots, a_n$, $2K = \sum a_k$, with the decision "Is there a subset of the integers whose sum is K?", we create a problem instance $\mathcal{I}_{PUCTP}$ as follows:

- S consists of two vertices with $s_1 = s_2 = K$
- D consists of n vertices with $d_k = a_k$ for all $1 \leq k \leq n$. That is, each integer a_k in I_{IP} corresponds to a destination with capacity a_k.

Observe that the optimum solution to $\mathcal{I}_{PUCTP}$ uses n packets if and only if the answer to I_{IP} is *yes*.

□

Because of this result, finding an exact solution for our MMLRP domain is not appropriate. So we examine ways to efficiently find approximate solutions.

3.2 Upper Bound

A fundamental observation — which is used in many of our proofs — gives an upper bound, in practice, of the cost of a solution. We say "in practice" because it is possible to construct a solution with higher cost than this, but it is trivial to always be able to achieve the following upper bound for any instance.

To produce a constructive proof of the upper bound, we first show a simple greedy approximation algorithm (as Algorithm 1) capable of finding a solution with fewer than $n + m$ messages.

Lemma 5. *Every instance of MMLRP can be solved with fewer than $n + m$ packets.*

Proof. Since every iteration of the while loop in Algorithm 1 increases the number of zero-valued sites by at least one while creating exactly one message in the solution, the resulting solution will have $n + m$ or fewer messages. However,

Algorithm 1 Greedy Approximation Algorithm

1: Initialize all $x_{i,j} = 0$
2: **while** there exists some $s_i > 0$ **do**
3: Find the largest s_i and largest d_j
4: $t \leftarrow \min(s_i, d_j)$
5: $x_{i,j} \leftarrow t$
6: $s_i \leftarrow s_i - t$
7: $d_j \leftarrow d_j - t$
8: **end while**

when the total supply and total capacities are equal, the last iteration is guaranteed to create two zero-valued sites along with one message. Further, when the total supply is less than the total capacity, the last iteration leaves at least one of the destinations non-zero. Thus, the number of generated messages must be $n + m - 1$ or less. □

In practice, finding the largest s_i and d_j can be done efficiently by using a heap data structure for the source and destination collections, which results in an $O(N \log N)$ time implementation, where $N = n + m$.

We note that to strictly achieve the upper bound result, the greedy approximation algorithm of Algorithm 1 can be run without sorting. It is enough to merely find an arbitrary s_i and an arbitrary d_j at step 3.

3.3 Approximating MMLRP

There are many tricks (heuristics) that we consider to improve the quality of a solution found by an approximation algorithm. The basic greedy algorithm has already been presented. We now investigate techniques to augment this strategy.

3.3.1 Matching Pairs

The first trick to approximating MMLRP is to look for i and j such that $s_i = d_j$, which we call a *pair*. We now show that it is never a bad decision to match up pairs. Hultberg and Cardoso present a proof of the following theorem for a maximum cardinality partition problem, which they show is equivalent to the PUCTP (see Theorem 3 in [7]). We present a proof directly using the transportation terminology.

Theorem 6. (Pairs) *Given an instance $\mathcal{I} = (S, D)$ with $s_i = d_j$ for some i and j, and a solution X for which $x_{i,j} < s_i$, there exists another solution X' with $x'_{i,j} = s_i$ such that $|X'| \leq |X|$.*

Proof. Since $s_i = d_j$, we know that $\sum_{k=1}^{m} x_{i,k} = \sum_{k=1}^{n} x_{k,j}$. If $x_{i,j} > 0$, we will remove this from consideration, but the sums remain equal. Let $S' = s'_1, \ldots, s'_{n'}$ be the non-zero entries of row i other than (i,j) (i.e. $S' = \{x_{i,k} | x_{i,k} > 0, 1 \leq k \leq m, k \neq j\}$). Similarly, let $D' = d'_1, \ldots, d'_{m'}$ be the non-zero entries of column j other than (i,j) (i.e. $D' = \{x_{k,j} | x_{k,j} > 0, 1 \leq k \leq n, k \neq i\}$). This subproblem, $\mathcal{I}' = (S', D')$, represents $n' + m'$ messages. There are n' messages coming from vertex i and there are m' messages bringing items to destination j.

Now we rearrange the transportation plan by assigning a message to carry s_i items from i to j (or augmenting the existing message $x_{i,j}$ to carry the full s_i amount if $x_{i,j} > 0$). This displaces m' messages which must be redirected to the n' destinations which have had some capacity freed up by the $x_{i,j}$ redirection.

Lemma 5 gives us an upper bound of $n' + m' - 1$ to solve $\mathcal{I}'$. So, replacing the $n' + m'$ messages removed from the original solution X, we have at most $n' + m' - 1$ plus the one message $x'_{i,j} = s_i$ (which is new if $x_{i,j} = 0$). Thus, we have not increased the size of the solution. □

This theorem implies that, if an instance has a pair, there is at least one optimal solution that utilizes this pair. It also shows that sub-optimal solutions can always be transformed, or possibly even improved, by "matching" up the pair.

3.3.2 Matching Couplets

We define a one-to-two matching as the existence of i, j, k such that either $s_i = d_j + d_k$ or $s_i + s_k = d_j$ which we call a *couplet.* A *triplet* refers to the existence of i, j, k, l such that either $s_i = d_j + d_k + d_l$ or $s_i + s_k + s_l = d_j$.

One would hope that matching up couplets would always be helpful. Alas, it is not always a good decision, as seen in the following example. Since each integer has a unique binary representation it is easy to see that this example has few couplets and no pairs. We describe three categories of sites, with the final instance $S = S_1 \cup S_2 \cup S_3$ and $D = D_1 \cup D_2$. First, there are three couplets:

$$S_1 = 2^0, 2^1, 2^2, 2^3, 2^4, 2^5$$
$$D_1 = 2^0 + 2^1, 2^2 + 2^3, 2^4 + 2^5$$

Second, one source that could fill these three destinations:

$$S_2 = 2^6 - 1$$

And third, a collection of triplets (each using one of the S_1 sources):

$$S_3 = 2^7, 2^8, 2^9, \ldots, 2^{18}$$
$$D_2 = 2^{18} + 2^{17} + 2^5,\ 2^{16} + 2^{15} + 2^4,\ 2^{14} + 2^{13} + 2^3,\ 2^{12} + 2^{11} + 2^2,\ 2^{10} + 2^9 + 2^1,\ 2^8 + 2^7 + 2^0$$

Observe that a good solution is to fill the three destinations in D_1 by sending three messages from the one source in S_2, then match up the six triplets in

S_3 and D_2 using 18 messages for a total of 21 messages. However, matching up the three couplets in S_1 and D_1 requires six messages, and the remaining destinations in D_2 requires 18 messages — from S_3 and splitting the S_2 source into three messages — for a total of 24 messages.

We can, however, say something about matching couplets. We either match up all of the couplet, or none.

Theorem 7. (All-or-Nothing) *Given an instance $\mathcal{I} = (S, D)$ with $s_i = d_j + d_k$ for some i, j, k, and a solution X for which $0 < x_{i,j} + x_{i,k} < s_i$, there exists another solution X' with $x'_{i,j} + x'_{i,k} = s_i$ such that $|X'| \leq |X|$.*

Proof. Without loss of generality, we assume $d_j \leq d_k$. The arguments are different depending upon the quantity of items going from i to destinations other than j and k. We consider the following cases:

Case 1: $x_{i,j}+x_{i,k} \geq d_j$. We ensure that destination j receives only items from i by swapping with destination k if necessary. Now we can remove destination j from the problem and subtract d_j from s_i to form a subproblem $\mathcal{I}'$. Note that a solution to $\mathcal{I}'$ leads immediately to a solution to $\mathcal{I}$. Also, Theorem 6 applies to $\mathcal{I}'$ giving the claimed result.

Case 2: $x_{i,j} + x_{i,k} < d_j$. If $x_{i,k} > 0$, then we swap the source i items with other items at destination j, thereby ensuring that $x_{i,k} = 0$. Note that we may have to "break" an item in destination j in order to accommodate this swap, thus creating an additional message. But this is compensated for by the merging of the source i items in destination j.

The argument is now similar to that used in Theorem 6. The difference is that after we pull out the $s_i - x_{i,j}$ source i items from their destinations and then find a solution to the subproblem using no more than $n' + m' - 1$ messages, we have two destinations to fill. But since we already have some of the source i items going to destination j, we merely increase $x_{i,j}$ to d_j — which adds zero messages to the overall solution — and use one message to transport the d_k items from source i to destination k. Thus our claim is achieved.

□

3.3.3 Double Split

One generalization on the All-or-Nothing theorem, is to have two sources splitting among two destinations in the following *double split* theorem.

Theorem 8. *Given an instance $\mathcal{I} = (S, D)$, and a solution X such that there exists $i, j \in S$ and $k, l \in D$ where $x_{i,k} > 0$, $x_{i,l} > 0$, $x_{j,k} > 0$, and $x_{j,l} > 0$, there exists another solution X' such that $|X'| < |X|$.*

Proof. Without loss of generality, let $x_{i,k}$ be a smallest of the four matrix entries. Then, $x_{j,l} \geq x_{i,k}$, and a swap can be done, merging both of the source i messages to destination l, to form X'. Specifically, $x'_{i,l} = x_{i,k} + x_{i,l}$, $x'_{i,k} = 0$, $x'_{j,k} = x_{j,k} + x_{i,k}$, and $x'_{j,l} = x_{j,l} - x_{i,k}$. Since $x'_{i,k}$ is zero, $|X'| < |X|$. □

This theorem implies that such a scenario will not exist in any optimal solution.

4 Improvements to the Greedy Approximation Algorithm

We can employ tricks described in the previous section to improve the quality of the resulting solution, at a cost of increased computational requirements. If we let $N = n + m$ be the problem size, the greedy algorithm described in Algorithm 1 can be implemented in $O(N \log N)$ time. If the search for the largest s_i and d_j is replaced with an arbitrary source and destination — thereby removing the pre-processing step of sorting — the complexity is $O(N)$ time.

We can make use of the Pairs theorem [Theorem 6] to potentially improve the resulting solution. There are two places this can be done in the greedy algorithm: as a pre-processing step before entering the while loop, and as a step inside the while loop. This modification is shown in Algorithm 2 with line 2 as the pre-processing step and line 9 is the pair removal step inside the loop. The advantage to looking for pairs inside the while loop is that the greedy algorithm may "create" pairs as part of its normal operation, and this will be discovered by the pairs search inside the loop. Note that the "created" pairs may be originally part of a couplet, a triplet, or some higher-order matching (i.e. one-to-Z, where $Z > 2$). However, looking for pairs inside the while loop does not necessarily find all couplets or higher-order matchings.

The pre-processing pairs search can be done in linear time on a sorted list, so this does not increase the running time analysis of the $O(N \log N)$-time greedy algorithm. Search for pairs inside the loop can be done in $O(\log N)$ time per iteration since only the "new" s_i or d_j, created at lines 6 or 7 of Algorithm 1, could possibly match up exactly with another site. Thus, the overall running time of the algorithm with the pairs search inside the while loop remains $O(N \log N)$.

Even though couplets are not guaranteed to produce optimum solutions, they do produce improved solutions in most situations (as we will see later). A detailed description of the implementation of couplets is given in Algorithm 3.

The running time of Algorithm 3 is dominated by the pre-processing couplets phase. An implementation of the loop at line 4, for example, may start with j and k at the ends and move inward until either $s_i = d_j + d_k$ or j and k cross. Doing this for every i results in an $O(N^2)$ total running time for this loop.

Of course, this technique can be extended to arbitrary "depth" by searching for triplets, one-to-four matchings, two-to-two matchings, etc.

Algorithm 2 Greedy Approximation Algorithm With Pairs

Initialize all $x_{i,j} = 0$
Remove all pairs $s_i = d_j$ † // pre-processing pairs step
while there exists some $s_i > 0$ **do**
  Find the largest s_i and largest d_j
  $t \leftarrow \min(s_i, d_j)$
  $x_{i,j} \leftarrow t$
  $s_i \leftarrow s_i - t$
  $d_j \leftarrow d_j - t$
  Removing any pairs created with the new s_i or d_j
end while

†This step involves setting $x_{i,j}$ to s_i (or d_j) then removing the sites, which is paramount to setting the site amount (supply or capacity) to 0.

Algorithm 3 Greedy Approximation Algorithm With Couplets

Initialize all $x_{i,j} = 0$
Remove all pairs $s_i = d_j$ † // pre-processing pairs step
  // pre-processing couplets
for all couplets $s_i = d_j + d_k$ **do**
  $x_{i,j} \leftarrow d_j$
  $x_{i,k} \leftarrow d_k$
  $s_i \leftarrow d_j \leftarrow d_k \leftarrow 0$
end for
for all couplets $s_i + s_j = d_k$ **do**
  $x_{i,k} \leftarrow s_i$
  $x_{j,k} \leftarrow s_j$
  $s_i \leftarrow s_j \leftarrow d_k \leftarrow 0$
end for
  // main greedy algorithm
while there exists some $s_i > 0$ **do**
  Find the largest s_i and largest d_j
  $t \leftarrow \min(s_i, d_j)$
  $x_{i,j} \leftarrow t$
  $s_i \leftarrow s_i - t$
  $d_j \leftarrow d_j - t$
  Removing any pairs created with the new s_i or d_j
end while

†This step involves setting $x_{i,j}$ to s_i (or d_j) then removing the sites, which is paramount to setting the site amount (supply or capacity) to 0.

5 An Example Application

The UK Meteorological Office (UKMO) is the UK national weather service. It is recognized as one of the world's leading providers of weather-related services and is at the forefront of international research in meteorological modeling.

Simplistically, meteorological modeling is the numerical solution of a number of partial differential equations which represent the behavior of the atmosphere. As these equations cannot be solved analytically, numerical approximations have to be calculated. Grid point codes discretize the underlying partial differential equations and obtain solutions on a discrete grid by progressing forwards in time, in time-steps. Such models can be split into two parts; the dynamics which keep the various fields in dynamical equilibrium (for example temperature and pressure), and the physics which parameterize the physical aspects of the atmosphere (for example solar heating and rainfall).

The UKMO's main modeling system, the Unified Model (UM), is a suite of grid point modeling codes used for both operational Numerical Weather Prediction (NWP) and Climate prediction. The UM is a large (around 1 million lines of Fortran code), powerful and flexible system. The UM can be run in either global or limited area configurations, and at many different resolutions.

Table 1 presents some of the most commonly used atmospheric resolutions. In the table, Grid is the distance between grid points in latitude and longitude and the Dimensions are the number of grid points in latitude, longitude and finally the number of levels. As there are a number of levels, a latitude and longitude position actually has a column of grid points we call a *grid column*.

Application	Grid (Km)	Dimensions	Time-step (mins)
Global Forecast	60	432x325x30L	20
UK Mesoscale	12	146x182x38L	5
Atmospheric Climate	270	96x73x19L	30

Table 1: Common UM configurations

Weather forecasting and Climate prediction is one of the so called Grand Challenge problems. Such problems can always make use of all the compute power that is available on the most powerful current generation machines. The UKMO currently run the UM on two Cray T3E's, one with 880 processors and the other with 500 processors. With such a large number of processors available an efficient parallel implementation of the UM is essential.

A domain decomposition approach is usually employed in parallel forecast and climate models, where equal sized regions of the computational grid are allocated to processors. In the case of UKMO, these regions are rectangular in latitude and longitude and there are an equal number of processors and regions. If no load balancing is performed, then all processors will compute an equal number of grid columns (ignoring differences due to integer division) and typically, for every time-step, a particular processor will compute the same grid columns.

This initial partition is not an arbitrary choice; it is selected to give the best overall performance. There are two main reasons for this. Firstly, in the UM, an equal amount of computation is performed at all grid columns for the majority of the dynamics and for the most computationally costly routine in the physics (Long wave radiation). Therefore, an equal number of grid columns for each processor is load balanced for these cases. Secondly, the dynamics also requires communication between neighboring processors and rectangular domains help reduce the communication-to-computation ratio and allow for a simple communication structure.

In the physics routines, grid columns are usually independent of each other and are therefore very amenable to parallelization; unfortunately, for the majority of these routines, the computational cost of these grid columns can vary significantly, leading to load imbalance. The existence of load imbalance is well known [3, 9] and has been widely acknowledged as being one of the major factors in loss of performance, particularly when using large numbers of processors, for example see [1, 4, 8, 10].

The most computationally intensive load-imbalanced routine in the UM is the *short wave radiation* subroutine. Short wave radiation models the effects on the atmosphere of radiation from the sun. Each processor calculates which of its grid columns are in daylight (a relatively trivial computation) and performs computation at these grid columns. As the short wave radiation subroutine calculates the effects of radiation from the sun, grid columns that are in darkness do not need to participate in this computation.

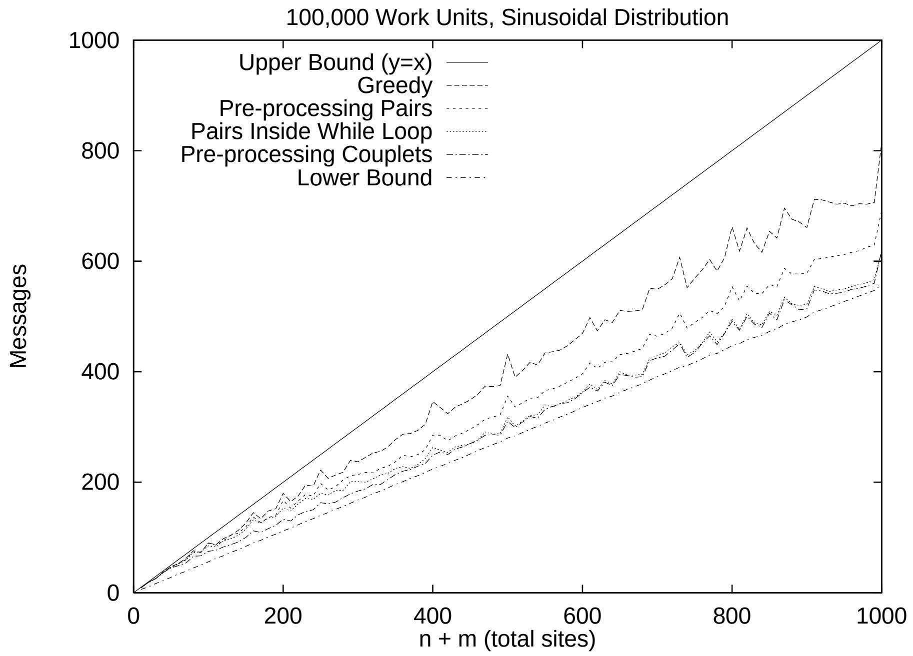
In a global model, half of the total grid columns will be in darkness (as half of the earth is always in darkness) and, for a reasonable number of processors, there will be some processors with all of their local grid columns in daylight and a number of processors with all of their local grid columns in darkness. As half of the total grid columns are in darkness and each processor has approximately the same number of local grid columns, the average number of sunlit grid columns per processor is half of its local grid columns. Since each of the sunlit grid columns requires the same amount of computation, processors with all of their local grid columns sunlit perform twice the average amount of work, giving a 50% load-imbalance. Redistributing the sunlit grid columns in the most efficient manner is the MMLRP.

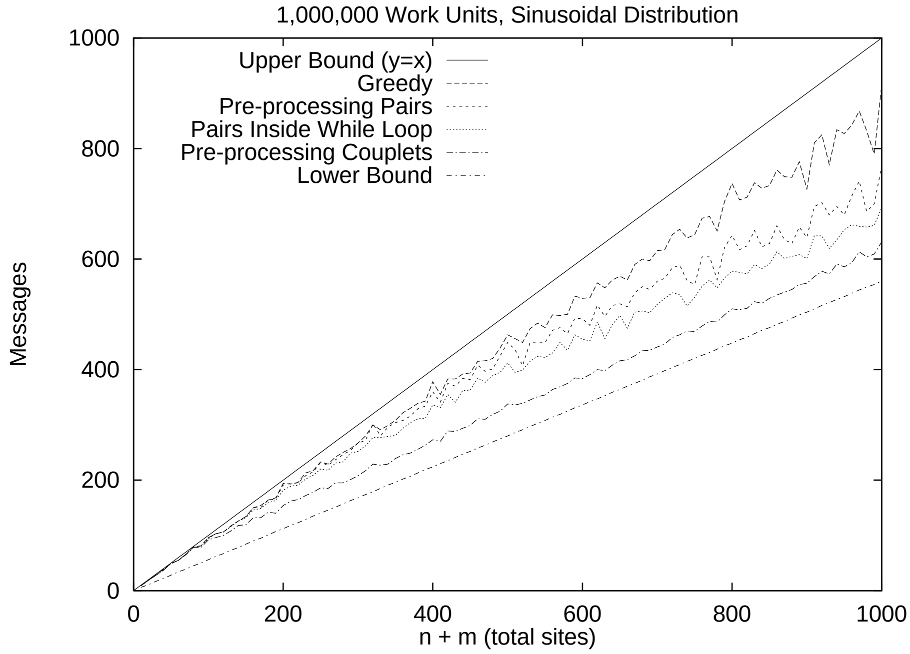
If short wave radiation were the only (or the single dominant) routine in the UM, or the majority of other routines had similar load characteristics then one could change the initial partition to reduce the load-imbalance observed. However, this is not the case and, as mentioned earlier, a different initial partition is preferable.

6 Experimental Results

The short wave radiation subroutine in the UKMO's UM typically has a load distribution that is close to a sinusoidal curve resulting from the pattern of sunlight hitting the earth's surface. So our simulation generates work units in a sinusoidal distribution among the sites (processors), and then runs the algorithms on the generated MMLRP and computes the cost of the transportation plan.

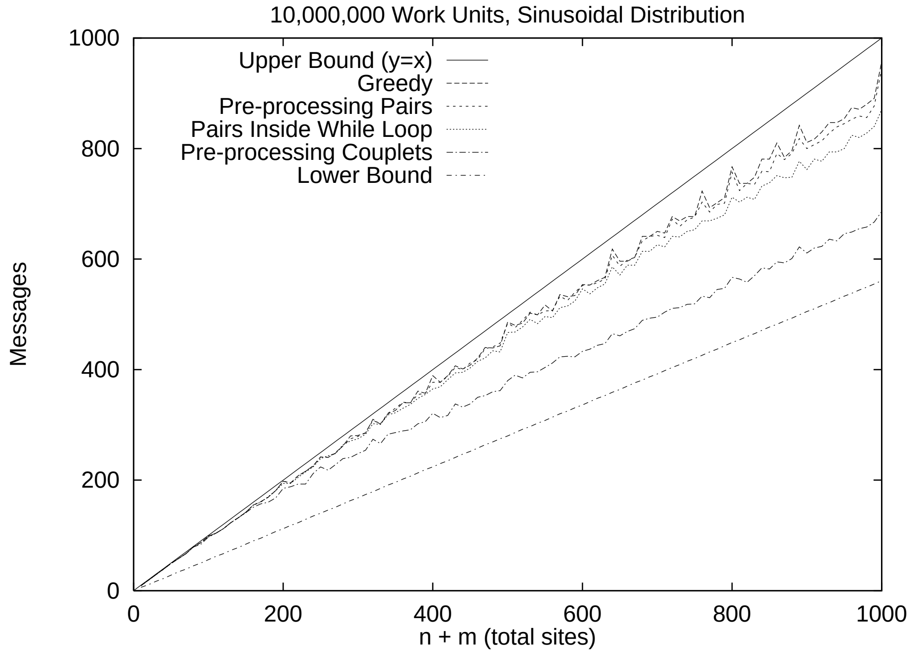
We ran the four algorithms, as presented above, to compare the quality of the solutions. These algorithms essentially build on the others so that we can describe them as an accumulation of enhancements. The algorithms are: the basic greedy approximation; greedy plus a pre-processing search for pairs; add to that a search for pairs inside the greedy algorithm's while loop; and finally, a pre-processing pairs following by a pre-processing couplets phase, then the basic greedy algorithm with the inside pairs search.





The upper and lower bounds are included as $n + m$ and $\max(n, m)$ as computed for each data point. They are not merely computed based on the sinusoidal distribution. These two approaches differ as the former does not include sites (processors) which are already balanced (i.e: are neither overloaded or underloaded). The latter assumes all processors are either overloaded or underloaded.

Since the number of work units affects the frequency of pairs and couplets in the input instance, we ran several different trials with differing numbers of work units. We ran one trial with 100,000 units, which is comparable to the the Global Forecast resolution (see Table 1). The second and third trials, of 1,000,000 units and 10,000,000 units respectively, demonstrate the potential impact of future resolutions.



7 Conclusions

7.1 Our contribution

We have given direct proofs for the $\mathcal{NP}$-completeness of PUCTP and the pairs theorem. This pairs theorem can be used to guide an algorithm along with couplets, which work well in practice.

We have shown that matching up couplets can produce a suboptimal solution, but experimentally it appears to be almost always helpful. We have also shown that with couplets, it makes sense to either match up completely or to deliver none between the three couplet sites.

As a result of this work the pairs algorithm (both pre-processing and inside the while loop) is now being used operationally by the UKMO for the short wave radiation routine. Load balancing this routine using the above algorithm has resulted in a 40% reduction in the routine's execution time compared with the default distribution [5].

7.2 Open Questions

There are many theoretical questions that remain unanswered. Is the PUCTP hard to approximate arbitrarily close? Or is there a polynomial-time approximate scheme?

Experimentally, it would be interesting to compare the computation time required for these algorithms to the actual communication cost savings for different environments such as Ethernet, or even across the Internet. It would also be interesting to determine how these algorithms perform on different initial load distributions such as Gaussian, and even on a random distribution.

Acknowledgments

The first author acknowledges the support of the EPSRC Visiting Fellowship GR/N05321. The second author acknowledges the funding support of the UK Meteorological Office.

References

1. P. Burton and A. Dickinson. Parallelising the unified model for the cray t3e. In G.-R. Hoffman and N. Kreitz, editors, *Proc. 7th Workshop on the Use of Parallel Processors in Meteorology*, pages 68–82. World Scientific, 1996.
2. G. B. Dantzig and M. N. Thapa. *Linear Programming.* Springer, 1997.
3. D. Dent et al. The IFS model performance measurements. In G.-R. Hoffman and N. Kreitz, editors, *Proc. 6th Workshop on the Use of Parallel Processors in Meteorology*, pages 352–369. World Scientific, 1994.
4. K. Eerola et al. A parallel version of the hirlam forecast model: strategy and results. In G.-R. Hoffman and N. Kreitz, editors, *Proc. 7th Workshop on the Use of Parallel Processors in Meteorology*, pages 134–143. World Scientific, 1996.
5. R. W. Ford and P. M. Burton. Load balancing physics routines. In W. Zwieflhofer and N. Kreitz, editors, *Proc. 8th Workshop on the Use of Parallel Processors in Meteorology*, pages 147–159. World Scientific, 1998.
6. M. Garey and D. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.
7. T. H. Hultberg and D. M. Cardoso. The teacher assignment problem: A special case of the fixed charge transportation problem. *European Journal of Operational Research*, 101(3):463–474, 1997.
8. K. M. and U. Schättler. REMO - implementation of a parallel version of a regional climate model. In G.-R. Hoffman and N. Kreitz, editors, *Proc. 7th Workshop on the Use of Parallel Processors in Meteorology*, pages 144–154. World Scientific, 1996.
9. J. G. Michalakes. Analysis of workload and load balancing issues in the NCAR community climate model. Technical report anl/mcs-tm-144, Argonne National Laboratory, Argonne, Illinois, 1991.
10. J. G. Michalakes and R. S. Nanjundiah. Computational load in model physics of the parallel NCAR community climate model. Technical report anl/mcs-tm-186, Argonne National Laboratory, Argonne, Illinois, 1994.
11. G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley, 1988.