DISTRIBUTED GENETIC ALGORITHMS FOR PARTITIONING UNIFORM GRIDS

By

Ioannis T. Christou

A dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN – MADISON

1996

Abstract

Ioannis T. Christou

UNIVERSITY OF WISCONSIN - MADISON, 1996

Under the supervision of Robert R. Meyer

In this thesis the author presents a new method for partitioning general large uniform 5-point grids into sub-domains of given areas having minimum total perimeter. For applications in scientific computing in parallel environments, this problem corresponds to minimizing the communication overhead between processors while observing load balancing constraints dictated by the speed of each individual processor. For a large class of grid shapes it is shown that the partition produced by this method is asymptotically optimal as the problem parameters grow to infinity. A new distributed Genetic Algorithm based on this decomposition theory significantly outperforms other well-known methods such as the spectral bisection (or quadrisection) methods and the geometric mesh partitioner.

Acknowledgments

To my parents, for their unlimited love and support throughout all these years.

Throughout my entire life my family has been the single source of love and true devotion that I knew I could always count on. To them I dedicate this thesis, hoping that one day they will realize how much I love them.

To my advisor, Prof. Robert R. Meyer, go my deepest thanks for guiding me in this research and sharing with me his valuable insights and mathematical thinking. Working with him in this relaxed friendly environment was a great pleasure.

I would like to thank the members of my Ph.D. committee, professors Olvi Mangasarian, Michael Ferris, John Strikwerda, and Paul Terwilliger (professors Mangasarian and Ferris also served as readers of this thesis). Whatever optimization background I now have, I certainly owe it to the CPO faculty at the Computer Sciences Dept. at the University of Wisconsin at Madison who always made teaching a friendly and exciting experience.

I would also like to acknowledge Yannis Schoinas, Dionysis Pnevmatikatos, Babak Falsafi, Andreas Moshovos and Armand Zakarian for many fruitful and fun discussions.

> Now when He was asked by the Pharisees when the kingdom of God would come, He answered them and said, "The kingdom of

God does not come with observation; Nor will they say "See here!" or "See there!". For indeed, the kingdom of God is within you."

Luke:ch. 17, 20–21

Contents

Chapter 1

Introduction

1.1 Problem Formulation

This thesis is concerned with the Minimum Perimeter problem (MP), a class of NP-hard problems that arise in scientific computations and engineering in parallel computing environments. In the solution of finite difference schemes ([Str89]), or the simulation of molecule behavior in Chemical Engineering or in edge detection in image processing ([Sch89]) and computer vision, or in the solution of max-flow problems over graphs with a grid structure using preflow-push [AMO93], one must perform a series of computations over a domain consisting of grid cells; the update of each cell requires its current value as well as the values of its immediate four neighbors, namely, its northern, southern, eastern and western neighbors. The name *5-point uniform grid* refers to any such computation (see figure 1). Often, some area of the domain is refined to a more detailed level of granularity to obtain better precision in this part of the grid. The grid is then no longer uniform. Even though the theory and main results of this thesis are concerned with the uniform case, many aspects are easily extended to the most general, non-uniform case as well.

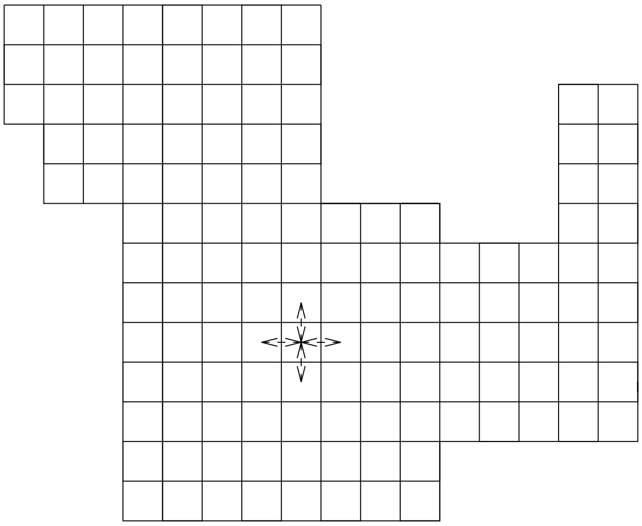

Figure 1: 5-point Uniform Grid Computation

In order to perform such 5-point computations over a discretized domain on a distributed memory parallel computer (like the Connection Machine CM-5 [Thi91] or a network of high-performance workstations) the computational load should be balanced across processors in a way that minimizes interprocessor communication. This communication will occur at the common boundaries of the regions that each processor will occupy. It is therefore necessary to partition the grid in such a way so as to incur as small a total perimeter of the partition as possible. As the parallel processing paradigm shifts towards networks of workstations where the communication delays can be very high compared to the processing speed of the machines, it becomes more and more important that high quality partitions of the given domain among the available processors can be found efficiently. An illustration of the network assignment nature of the problem is shown in figure 2. It is assumed that processor i will be assigned a workload of a_i cells. (In a homogeneous processor environment the a_i will be chosen to be as nearly equal as possible.) Processor p_i represents a supply node of supply a_i. Each grid cell is a demand node of demand 1, corresponding to the assignment of its task to exactly one processor, therefore the goal is to find a feasible assignment that minimizes the total perimeter. A provably

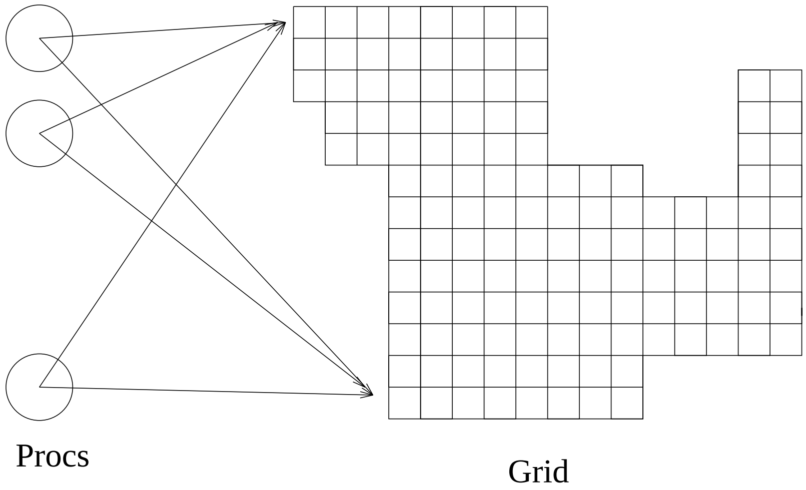

Figure 2: Network Assignment Formulation of MP

optimal partition of a 7×7 grid equi-partitioned among 7 processors is shown in figure 3.

In its most general form, the Minimum Perimeter problem MP($\mathcal{G}$, P) can be stated as follows: *Given a Grid* $\mathcal{G}$ *of unit cells, a number of processors* P*, and an associated load* a_i *for each processor, find an assignment of the grid cells to the processors so that the perimeter of the partition is minimized while observing the load balancing constraint that processor* p *is assigned exactly* a_p

1	1	1	2	2	3	3
1	1	2	2	2	3	3
1	1	2	2	3	3	3
4	4	4	4	5	5	5
4	4	4	5	5	5	5
6	6	6	6	7	7	7
6	6	6	7	7	7	7

Figure 3: 7×7 grid partitioned among 7 processors

cells. The perimeter of a partition is the sum of the lengths of the boundaries of the regions that each processor occupies , while the load of each processor is the area of the region it occupies. By considering the graph of the grid, where each grid cell corresponds to a vertex and each border between two neighboring cells corresponds to an edge between the corresponding vertices, the problem becomes a Graph Partitioning problem, and even though it is restricted in the class of uniform 5-point grids, it remains NP-hard (see [Yac93]). MP can easily be formulated as a Quadratic Assignment problem ([PRW93]), with $|\mathcal{G}|P$ binary variables and $|\mathcal{G}| + P$ constraints. Letting $\mathcal{I}$ denote the set of pairs of adjacent cells, and a_p the area for processor p, the QAP formulation is as follows:

$$\min. \sum_{i,j \in \mathcal{G}} \sum_{\substack{p,p'=1 \\ p \neq p'}}^{P} c_{ij} x_i^p x_j^{p'} \tag{1}$$

$$s.t. \begin{cases} \sum_{i \in \mathcal{G}} x_i^p = a_p & p = 1 \ldots P \\ \sum_{p=1}^{P} x_i^p = 1 & i \in \mathcal{G} \\ x_i^p \in \mathbf{B} = \{\mathbf{0}, \mathbf{1}\} \end{cases}$$

$$\text{where } c_{ij} = \begin{cases} 1 & \text{if } (i,j) \in \mathcal{I} \\ 0 & \text{else} \end{cases}$$

The objective of this optimization problem is a sum of quadratic terms of binary variables, while the constraints are network constraints as shown in figure 2.

At the expense of introducing more variables and constraints, we can reformulate the problem as a mixed linear integer program:

$$\min. \sum_{i,j \in \mathcal{G}} \psi_{ij} \tag{2}$$

$$s.t. \begin{cases} \sum_{i \in \mathcal{G}} x_i^p = a_p & p = 1 \ldots P \\ \sum_{p=1}^{P} x_i^p = 1 & i \in \mathcal{G} \\ \psi_{ij} \geq x_i^p + x_j^{p'} - 1 & (i,j) \in \mathcal{I}, \;\; p \neq p' = 1 \ldots P \\ \psi_{ij} \in \mathbb{R} \\ x_i^p \in \mathbf{B} = \{\mathbf{0}, \mathbf{1}\} \end{cases}$$

As the size of the grid increases, the number of binary variables needed in any of the above formulations renders the problem intractable by means of classical Branch & Bound methods ([NW85]), and there is very little hope of solving exactly the problem as formulated above. Instead, we use a high level approach, taking into account the geometric aspects of the problem, to develop a theory that enables us to construct high quality solutions very efficiently. For large subclasses, this theory has the property that as the problem parameters grow to infinity, the resulting partitions become asymptotically optimal in the sense that their relative distance from a computable lower bound approaches zero. We focus our attention on the equi-partitioning case, where, assuming equal speeds among the processors, the balancing constraints become equi-partitioning constraints, i.e. we require that the size of any two components differ by no more than 1.

1.2 Applications and Motivation

As already stated, the main motivation for the MP comes from the solution of PDEs using finite difference schemes in parallel computing environments. Here we discuss another application that illustrates the need for solving MP efficiently.

In high temperature super-conductors, studying a single magnetic flux line can lead to a graph model citeChen that is shown in figure 4. In this graph, a “forward” edge is an edge whose direction is left to right, and a “backward” edge is one connected right to left. The capacities of the "forward" edges are random integers uniformly distributed in the interval $[0 \ldots 4096]$ which model oxygen defects in the super-conductor that pin the flux line to the minimal cut (which in the physics literature is called the optimal path), and the "backward" arcs have infinite capacity and are there to guarantee a continuous cut since a physical flux line must be continuous. With this setup, the interesting properties of the graph are its max flow, and the minimum cut associated with it which is in a one-to-one correspondence with a physical flux line.

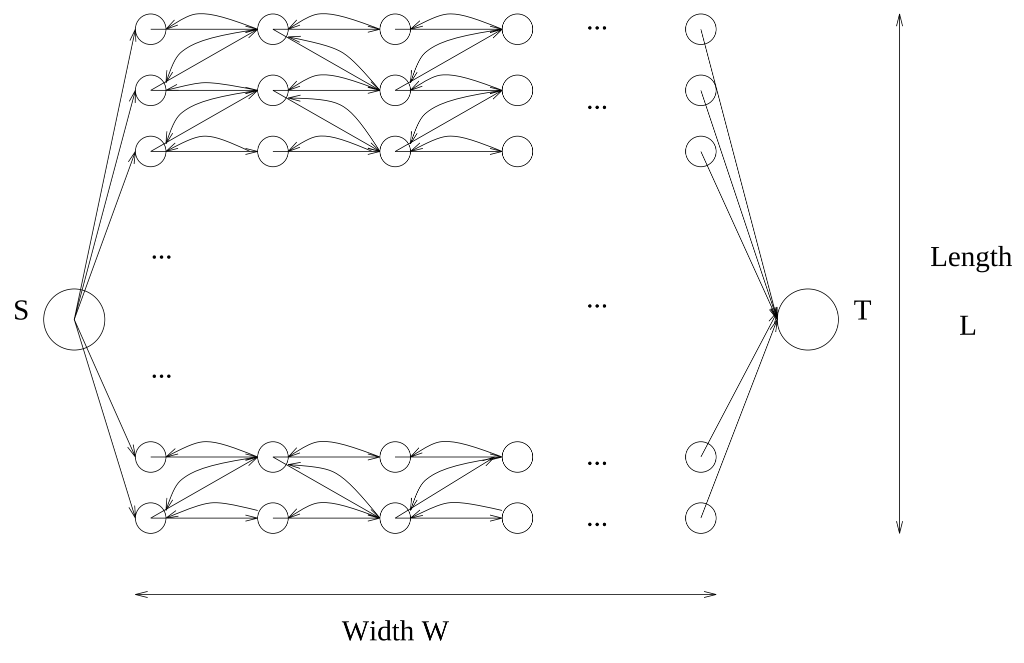

Figure 4: Flux Lines in High Temperature Super-conductors

However, the study of the asymptotic behavior of this flux line, needs extremely large graph sizes: the width W of the graph should be around 250 and the length of the graph should be between ten thousand and one hundred thousand, resulting in graphs of up to 25 million nodes. Very few machines today have enough memory to store all of the graph in main memory. Thus the need for computing the max-flow problem in parallel becomes crucial not only because of the potential speed-up in response time, but also because simply, no single machine has enough memory to store the problem (let alone solve it).

Preflow push is an algorithm suitable for computing max-flow in a graph that not only has an excellent theoretical running time but also offers the great advantage of being amenable to parallelism because it essentially performs local computations; each iteration selects a node from a queue of active nodes (nodes with excess flow) and attempts to distribute the excess flow to neighbor nodes that are closer to the destination. Observing the structure of the graph, one can easily see that essentially this is a 5-point computation where the update of the values of a node (its excess flow and its distance from the destination) needs only the values of its immediate neighbors. Therefore the max-flow problem can be solved in parallel, with communication among processors occuring only at the borders of the areas of the graph each processor occupies (there is one drawback though: initially, only nodes adjacent to the source have excess flow, and so processors having no nodes adjacent to the source will have to remain idle until another processor pushes some flow to one of their nodes; this means that a partitioning scheme that leaves many processors waiting idle in the beginning of the computation even though it minimizes communication might not enhance overall performance).

The above max-flow problem makes evident the need for processing in parallel such big graphs. This need is dictated by memory requirements since for even a relatively small test problem of size $W = 250$ and $L = 2850$ preflow push requires about 127MB of memory, more than the physical memory most workstations come equipped with. Therefore, splitting the problem among a network of workstations is a good choice. To enhance performance then, a good partitioning of the nodes of the graph is needed. This thesis presents a methodology for computing efficiently such partitions.

1.3 Outline of the DGA Approach

Before we present a brief survey of the literature in the area of graph partitioning, we outline our approach to solving the MP. Our method takes into account the geometric nature of the problem which allows us to approximately identify the shapes optimal partitions should have. This knowledge turns out to be the key to the stripe (and snake) decomposition theory that we develop in the next chapter, which shows that near-rectangular grids of certain classes can be partitioned among a large number of processors asymptotically optimally. Snake decomposition accepts as input a partition of the rows of the grid, and fills each stripe with processor indices obeying load balancing constraints. To choose among a large number of input candidate row-partitions of the grid, we use a distributed, fully asynchronous Genetic Algorithm which uses snake decomposition as the fitness function to evaluate and rank each individual in the population. Our GA (called DGA), employs the island model of computation but includes some new data structures (and an aging mechanism) for handling migration, and to avoid premature convergence phenomena (see chapter 4 for an extensive discussion on the workings of this algorithm). The results it has produced (presented in chapter 5) are in general superior to the other methods we have tested for non-rectangular grids.

1.4 Related Work

As noted before, MP is a graph partitioning problem (each grid cell representing a vertex and each boundary between two cells representing an edge between the two cells). In the remainder of this chapter, we review algorithms that have been proposed before for solving MP. Many of the well established methods, like the spectral methods, the Kernighan-Lin method or the geometric mesh partitioner , work on arbitrary graphs; however, many of them also have the disadvantage of requiring the number of partitions to be a power of two.

1.4.1 The Kernighan-Lin Heuristic

One of the most famous algorithms for graph partitioning, the K-L heuristic [KL70] is a very well established method for general graph partitioning that is used in many modern codes as a post-processing routine. It requires an arbitrary initial partition of the graph in two (balanced) sets and then until no more improvement can be made, it exchanges pairs of nodes between the two sets so as to improve the total cost of the partition. In order to effectively do these swaps, it maintains some heap data structures to help sort the gains of the nodes; the complexity of performing a swap (always the best swap is performed, i.e. the swap that maximizes the cost improvement of the partition) is then shown to be $O(n^2 \log(n))$ where n is the number of vertices in the graph. For an unweighted version of the graph partitioning problem, the minimum number of K-L iterations is $O(|E|)$ where E is the set of edges of the graph. The K-L algorithm in pseudo-code is as follows:

```
procedure K-L(G(V,E), CurrentPartition)
begin
do
  BestPartition = CurrentPartition;
  S1 = {v | v in Partition1};
  S2 = {v | v in Partition2};
  for each v in V do
    pref[v] = ComputePreferenceValue(v,S1,S2);
  endfor;
  while S1!={} and S2!={} do
    v = argmax(pref[v]) over all v in S1;
```

```
    insert(v, CurrentPartition2);
    delete(v, S1);
    for each w neighboring v do
      pref[w] = UpdatePreferenceValue(w,v);
    endfor;
    v = argmax(pref[v]) over all v in S2;
    insert(v, CurrentPartition1);
    delete(v, S2);
    for each w neighboring v do
      pref[w] = UpdatePreferenceValue(w,v);
    endfor;
    if cost(CurrentPartition) < cost(BestPartition)
      BestPartition = CurrentPartition;
    endif;
  endwhile;
  CurrentPartition = BestPartition;
until no_improvement;
end;
```

In general, K-L is a fast and efficient local refinement technique that can improve on the partitions found by other algorithms. However, it does need a relatively good partition to begin with. Extensions to the K-L algorithm are presented in [HL93] where the authors show how a generalized K-L heuristic (GKL) can partition a graph among an arbitrary number of nodes.

1.4.2 Spectral Methods

The spectral bisection algorithm for sparse graphs was proposed in [PSL90], and attempts to partition an arbitrary graph into two components of equal size while minimizing the cut edges of the partition. The idea behind the partitioning scheme traces back to the work in algebraic graph theory done by Fiedler ([Fie73] which relates the second smallest eigenvalue and corresponding eigenvector of the Laplacian matrix of a graph $\mathcal{G}$ to the edge and vertex connectivities of the graph. The Laplacian matrix of a graph $\mathcal{G}(V, E)$ with n nodes is defined as follows:

$$L_{ij} = \begin{cases} -1 & \text{if } (v_i, v_j) \in E \\ d_i & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

where d_i is the degree of node v_i. It is easy to check that $L = D - A$ where D is a diagonal matrix with diagonal elements the node degrees of the graph and A is the adjacency matrix of the graph. Essentially, spectral bisection attempts to solve the discrete optimization problem

$$\min.\frac{1}{4}x'Lx \tag{3}$$

$$s.t. \begin{cases} x'e = 0 \\ x_i = \pm 1 \end{cases}$$

which is equivalent to solving the general graph partitioning problem exactly. The constraints simply require the partition to be balanced and to indicate to which partition each node belongs (all x_is of one sign belong to one partition, and all x_is of the opposite sign, to the other partition). The objective counts cut edges. To see this observe that $\frac{1}{4}\sum_{(i,j)\in E}(x_i - x_j)^2$ is the number of edges crossing between the two partitions because $(x_i - x_j)^2$ contributes nothing to the sum if x_i and x_j have the same sign, and contributes 4 otherwise. Now by writing

$$\sum_{(i,j)\in E}(x_i - x_j)^2 = \sum_{(i,j)\in E}(x_i^2 + x_j^2) - 2\sum_{(i,j)\in E} x_i x_j = \sum_{(i,j)\in E} 2 - x'Ax = x'Dx - x'Ax$$

we get that $\frac{1}{4}\sum_{(i,j)\in E}(x_i - x_j)^2 = \frac{1}{4}x'Lx$.

Since graph partitioning is NP-complete, there is little hope of solving (3) exactly, and thus spectral bisection *relaxes* the problem by solving its continuous counterpart with the integrality constraints replaced by the constraint $x'x = n$. It is exactly this relaxation that is the key approximation in the application of spectral methods. As discussed in [HL95b], the relaxed problem has as a solution $x = \sqrt{n}u_2$ where u_2 is the normalized eigenvector corresponding to the second lowest eigenvalue λ_2 of L, and n is the cardinality of V, and this solution is unique if $\lambda_2 \neq \lambda_3$ where λ_3 is the third lowest eigenvalue of L. This is due to a theorem proved in [HL95b] stating that the matrix L is symmetric positive semi-definite, its normalized eigenvectors are pairwise orthogonal, and that the smallest eigenvalue of L is zero, with an associated normalized eigenvector $u_1 = \frac{1}{\sqrt{n}}e$.

So the solution of the relaxation of the discrete optimization problem (3) is reduced to an eigenvalue computation of the Laplacian matrix of the graph, which can be performed quite efficiently using Lanczos' algorithm or some other polynomial time algorithm. Then, the relaxed solution must be mapped back into the feasible region of the discrete problem. Spectral bisection does this by finding the median of the vector x and assigning all nodes with x_i less than the median to one partition ($x_i = -1$) and all the other nodes to the other partition ($x_i = +1$). This integer point represents the point satisfying the constraints in (3) that is closest to x, and the hope is that it is an optimal solution of (3) or near-optimal. The above theory also shows that a lower bound on the number of cut edges of the best separator of the graph $\mathcal{G}(V, E)$ is given by $\frac{n\lambda_2}{4}$. By using the eigenvectors of the Laplacian matrix of the graph (which are properties of the whole graph), this method makes use of global information about the graph, as opposed to other local refinement graph partitioning schemes like the Kernighan-Lin algorithm [KL70].

The spectral bisection algorithm and its extensions (recursive spectral bisection, or RSB for short, for partitioning an arbitrary graph into a number of processors that is a power of two, and recursive spectral quadrisection and octasection [HL95b] that generalize bisection by considering two or three eigenvectors of the Laplacian matrix of the graph) have received a lot of attention because of their demonstrated excellent performance. However, they have two potential drawbacks; the first is the need to construct the Laplacian matrix of the graph which in large problems can lead to memory problems. The second drawback is the fact that they cannot easily partition a graph into an arbitrary number of components. For this end, enough dummy nodes must be introduced, and the resulting partition is unlikely to be near-optimal (balancing issues might also arise). Furthermore, as the computational results show, these methods fail to find very good quality solutions on the class of large 5-point uniform grids by failing to take full advantage of the geometry and special properties of the grid.

The spectral quadrisection and octasection algorithms attempt to split a graph into four or eight components at once using the two or three eigenvectors corresponding to the two or three lowest eigenvalues of L (excluding the zero eigenvalue). Practical experience shows that on general meshes these methods perform better than RSB, but in our experience with domains that are uniform 5-point grids, RSB often works better (even though a bit slower) than its multi-dimensional counterparts.

1.4.3 Geometric Mesh Partitioning

In 1993, Miller et. al. proposed a new method for partitioning a d-dimensional mesh that used geometric information of the graph, namely the co-ordinates of its nodes in $\mathbb{R}^d$ to obtain provably good edge or vertex separators [MTTV93]. The idea underlying geometric partitioning is that once the co-ordinates of the nodes of a given graph are given, one can map the d-dimensional mesh onto a

$(d+1)$-dimensional space using a stereographic projection which will map all the nodes of the graph into the unit sphere in $\mathbb{R}^{d+1}$. Any node v in $\mathbb{R}^d$ is projected to this sphere at the point where the line passing through v and the north pole $(0, \ldots, 0, 1)$ and the sphere intersect. Then, as soon as the centerpoint of the projected nodes has been computed, a rotation of all the projected nodes about the origin is performed until the centerpoint aligns itself somewhere on the $(d+1)$-st axis. Then, the nodes on the surface of the sphere are dilated until the centerpoint becomes the origin. Then the method chooses a random circle on the unit sphere in $\mathbb{R}^{d+1}$, and maps it back to $\mathbb{R}^d$ by undoing the dilation, rotation, and stereographic projection operations. Then, all the original nodes in $\mathbb{R}^d$ which are located in the "neighborhood" of the projected circle form a vertex separator of the graph, from which one can compute an edge separator as well.

The most striking property of this method is that doesn't make use of edge information at all. Indeed, all that it requires to compute the vertex separator is the co-ordinates of the nodes of the graph. Its power comes from the fact that in most geometric meshes that arise from practical applications, nodes that are away from each other in $\mathbb{R}^d$, tend not to have edges connecting them. The authors define the notion of a k-ply neighborhood system as a set of n closed disks $(D_1, \ldots, D_n)$ in $\mathbb{R}^d$ such that no point in $\mathbb{R}^d$ is strictly interior to more than k of the disks. Then they show that if the mesh has a bounded aspect ratio, or is what they call an (a, k)-overlap graph, then a provably good vertex separator for the graph exists. An (a, k)-overlap graph for a neighborhood system $(D_1, \ldots, D_n)$ for which each disk's center is located at node v_i, is a graph with vertex set $(v_1, \ldots, v_n)$ and an edge between any two nodes v_i, v_j such that $D_i \cap aD_j \neq \emptyset$ and $D_j \cap aD_i \neq \emptyset$.

The geometric mesh partitioner attacks the problem of graph partitioning using the idea of exploiting the position of the nodes in space to extract "good" cuts, but requires geometric information about the graph that might not be available. Furthermore, generalizing it to partition meshes into an arbitrary number of components that is not a power of two is not straightforward .

1.4.4 Greedy Randomized Adaptive Search Procedures

Another method for partitioning an arbitrary weighted graph into two balanced partitions while minimizing the sum of the weights of the cut edges is presented in [LFE94]. This GRASP heuristic chooses one node at a time from a list of candidate nodes and alternates insertion of the chosen node between the two partitions. The nodes in the candidate list are the ones that maximize a gain function, which is simply the sum of the weights of the arcs from the given node to the nodes in the partition which will accept the node minus the sum of the weights of the arcs from the given node to the nodes in the other partition.

This GRASP heuristic, when implemented with appropriate data structures, has been shown to outperform the Kernighan-Lin heuristic. Even though the authors do not provide an extension to the general k-way partitioning, it is conceptually easy to design a GRASP heuristic for partitioning a graph among any number of components.

1.4.5 Fair Binary Recursive Decomposition

Fair Binary Recursive Decomposition (FBRD) is a method proposed by Crandall and Quinn in [CQ95] for the partitioning of uniform and non-uniform 5-point grids; their discussion is focused on rectilinear grids. They allow for different processor speeds which implies possible different area sizes for each partition. Their method divides the processors into two sets of processors so that the sum of the processor speeds in each list is as balanced as possible. Then, they divide the grid along the longest dimension of the (rectilinear) grid so that the resulting sub-grids can accommodate the processor sub-lists. Then, each of the sub-grids is partitioned among the processors in the processor list assigned to it recursively.

The method, although mainly focused on rectilinear 5-point grids (uniform or non-uniform), allows partitions among an arbitrary number of processors with different speeds. The main problem with FBRD is that for large numbers of processors, this technique may lead to inefficient partitions due to the fact that the individual components no longer look near-rectangular.

1.4.6 Genetic Algorithms Using Intelligent Structural Operators

GAs have been proposed before for solving the graph partitioning problem. In [vL91], the author uses a representation for a partition of a graph into k components that assigns one allele in the chromosome for each node in the graph. This means that the total length of the chromosome of each individual is at least $|V|$ where V is the node set of the graph. Simple crossover and mutation is likely to result in an unbalanced (and thus infeasible) partition, so a different strategy is used to create the offspring: first a component is selected from one of the parents, and is copied onto a temporary copy of the other parent. In this temporary copy, all the nodes that were originally assigned to the same component that was just copied but not copied over again, are unassigned, and assigned to another component using a repair strategy so as to maintain the balance of the partition. Similarly, a mutation is defined in this context as the exchange of two numbers of the coding. Finally, a variant of the K-L heuristic is applied to the offspring partition. The GA is run on a network of transputers, and the selection process follows a neighborhood model (instead of the traditional panmictic model) where individuals chose their mates from a neighborhood scheme that is implemented as a ring.

The experimental results showed the superiority of the GA versus K-L for partitioning a graph among any number of processors for small graphs (one randomly generated graph with 900 nodes and average node degree of 4 served as the test bed of the experiments reported in the paper). It is not clear how the algorithm will perform as the number of nodes increases (to the order of many thousands).

1.5 General Overview of the Remaining Chapters

In the next chapter, we present a theory that shows that asymptotically optimal partitions of any near-rectangular grid can be efficiently computed as long as the number of components is large enough. Chapter 3 then presents the snake decomposition algorithm in detail. In chapter 4 we discuss Genetic Algorithm issues in general, and then present DGA, the Distributed Genetic Algorithm we have developed for finding good inputs to the snake decomposition algorithm, and discuss its merits. Chapter 5 presents the computational results from the runs of DGA, as well as from other popular methods of graph partitioning. Finally, in the chapter entitled "Future Directions" we outline promising directions that research in this area might take.

Chapter 2

Asymptotically Optimal Solutions via Stripe Decomposition

2.1 Overview

In this chapter we present theoretical results about error bounds for large classes of domains. First, we present a computable lower bound on the perimeter of the equi-partition of *any uniform 5-point grid* among P processors, and then, we proceed to show that for rectangular domains, as long as the number of processors dominates the individual dimensions of the rectangle, partitions exist with a total perimeter whose relative distance from the lower bound goes to zero as the problem parameters tend to infinity. A very important aspect of these theorems is that the proofs are constructive, i.e. we provide a way to compute such solutions in polynomial time. Then, we extend the theorems to encompass more general grids (*irregular-boundary* grids) and show that as long as the grid contains a finite number of large rectangular areas to be partitioned among many processors, the above properties still hold, i.e. there exist partitions whose relative distance from the lower bound converges to zero as the problem parameters tend to infinity. These theoretical results will provide the basis of the algorithms to be discussed in the next chapters.

2.2 Lower Bounds and Optimal Configurations

As Yackel and Meyer have shown in [YM92a], for any given area $\mathcal{A}_p$, a lower bound on the perimeter of *any* configuration of $\mathcal{A}_p$ cells is given by

$$\Pi^*(\mathcal{A}_p) = 2\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil. \tag{4}$$

Furthermore, this lower bound is tight in the sense that there exists a non-empty library of configurations, called the *optimal shapes* for $\mathcal{A}_p$ that achieves this lower bound on the perimeter. These shapes can be generated using the following technique: start with a rectangle that has perimeter $\Pi^*(\mathcal{A}_p)$ and area at least $\mathcal{A}_p$. Iteratively remove the corner cells of this rectangle until the area of the remaining object is exactly $\mathcal{A}_p$. The remaining object is an optimal shape for $\mathcal{A}_p$. It turns out that all the optimal shapes for a given area size $\mathcal{A}_p$ can be constructed using this technique (see Yackel's PhD thesis [Yac93]). The optimal shapes for areas 5 and 7 (modulo reflections and rotations) are shown in figure 5.

All of these optimal shapes have a property called slice-convexity (which is

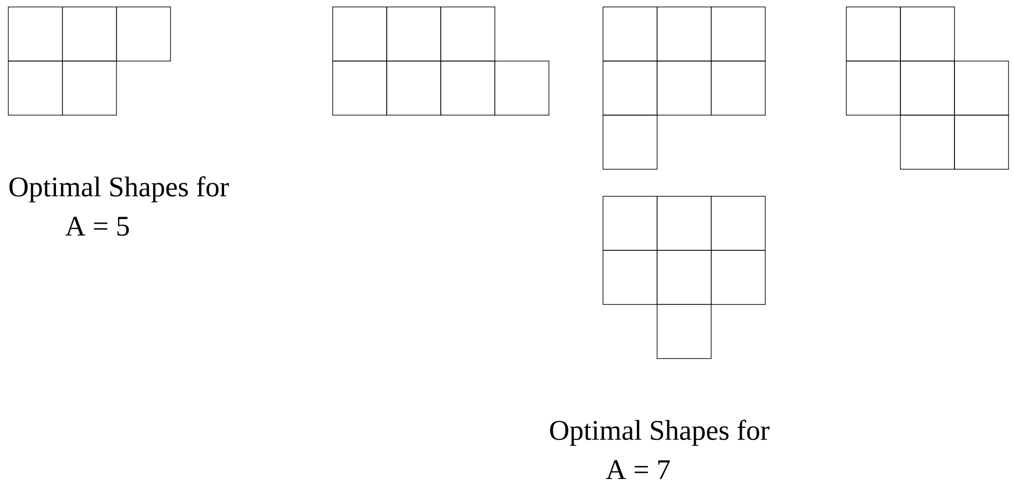

Figure 5: Optimal Shapes for Areas 5 and 7

the same as convexity in the "polyomino" literature), a consequence of which is the fact that the perimeter of any optimal shape is twice its semi-perimeter, where the semi-perimeter of any shape is the sum of the height and width of the smallest rectangle containing the shape. Many of these optimal shapes are rectangular blocks with a "fringe" attached to one of their sides [YM92b], so they can be characterized by three numbers, namely the dimensions of the rectangle h, w and the size of the fringe f. We denote such a near-rectangular shape by (h, w, f). In general, the number of such near-rectangular optimal shapes is of order $\mathcal{A}_p^{1/4}$ (see [YMC95]), but this does not encompass all possible minimum perimeter configurations. There is a lot of literature (see for example [Lin91, Mel94]) dealing with the generating function approach for developing expressions for the exact number of "convex polyominoes" with various properties.

These near-rectangular optimal configurations play a key role in the analysis below, and we prove a particular optimality test for these shapes:

Lemma 1 *Given an area $\mathcal{A}_p$ of unit cells, let $k := \left\lfloor\sqrt{\mathcal{A}_p}\right\rfloor$. Then*

- *if $k^2 = \mathcal{A}_p$ the shape $(k, k, 0)$ (a $k \times k$ rectangle) is an optimal shape and its perimeter is $4k$.*
- *if $k^2 < \mathcal{A}_p \leq k(k+1)$ then the shapes (k, k, f) and $(k+1, k-1, f')$ with $f \leq k$ and $f' \leq k+1$ are optimal shapes and their perimeter is $4k+2$.*
- *else if $k(k+1) < \mathcal{A}_p$ then the shapes $(k, k+1, f)$ and $(k+1, k, f)$ with $f \leq k$ are optimal and their perimeter is $4(k+1)$.*

Proof:

- Consider $k^2 = \mathcal{A}_p$. The lower bound on the perimeter of any configuration of $\mathcal{A}_p$ cells is $2\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil = 4k$. Since $(k, k, 0)$ has area $\mathcal{A}_p$ and perimeter $4k$ it is an optimal shape.
- Consider the case $k^2 < \mathcal{A}_p \leq k(k+1)$. Now the lower bound is $2\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil \geq 4\sqrt{\mathcal{A}_p} > 4k \Rightarrow \Pi^*(\mathcal{A}_p) \geq 4k+2$ (since the Π^* is always an even number). Now, (k, k, f) with $f = \mathcal{A}_p - k^2 \leq k$ has perimeter $2(2k+1)$, and so is an optimal shape, as is the shape $(k+1, k-1, f')$ with $f' = \mathcal{A}_p - k^2 + 1 \leq k+1$ because its perimeter is also $2(2k+1)$.
- Finally, assume that $k(k+1) < \mathcal{A}_p$. Then $2\sqrt{\mathcal{A}_p} = 2(k+r)$ with $r \in (0,1)$ and therefore $\Pi^*(\mathcal{A}_p) = 2\left\lceil 2k + 2r\right\rceil$ which is equal to $4(k+1)$ iff $r > \frac{1}{2}$.

But we have that

$$\sqrt{\mathcal{A}_p} = k + r \Rightarrow \mathcal{A}_p = k^2 + 2kr + r^2 > k^2 + k \Leftrightarrow$$

$$\mathcal{A}_p - k(k+1) = r^2 + 2kr - k > 0$$

If $r \leq \frac{1}{2}$ then $r^2 \leq \frac{1}{4}$ and we get $\mathcal{A}_p - k(k+1) \leq \frac{1}{4}$ which is a contradiction since $\mathcal{A}_p - k(k+1)$ is an integer greater than 0. So $r > \frac{1}{2}$, and therefore $\Pi^*(\mathcal{A}_p) = 4(k+1)$. The shapes $(k, k+1, f)$ and $(k+1, k, f)$ with $f = \mathcal{A}_p - k(k+1) < k+1 \Rightarrow f \leq k$ (since $(k+1)^2 > \mathcal{A}_p$) have perimeter equal to $4(k+1)$ and so are optimal configurations for $\mathcal{A}_p$.

∎

When P, the number of processors divides the total area A of the grid, in the equi-partitioning case each processor is assigned $\frac{A}{P}$ cells. However, when P does not divide A, some processors will be assigned one more cell than others (to keep the loads as balanced as possible). Assuming that P_1 processors will get a lower load A_1 and that the rest $P_2 := P - P_1$ processors will get a heavier load $A_2 := A_1 + 1$, from the equations

$$A_1 P_1 + A_2 P_2 = A$$

$$P_1 + P_2 = P$$

we get that

$$P_1 A_1 + (P - P_1)(A_1 + 1) = A \implies P A_1 + P_2 = A.$$

Therefore,

$$\text{for } i = 1 \ldots P_1, \quad a_i = A_1 = A \div P = \left\lfloor \frac{A}{P} \right\rfloor$$

$$P_1 = P - A \mod P$$

$$\text{and for } i = P_1 + 1 \ldots P \quad a_i = A_2 = A \div P + 1 = \left\lceil \frac{A}{P} \right\rceil$$

$$P_2 = A \mod P$$

.

A lower bound on the optimal value of the MP($\mathcal{G}, P$) (equi-partition of a uniform 5-point grid of total area A among P processors) is then given by

$$L(A,P) = \begin{cases} P\Pi^*(\frac{A}{P}) & \text{if } A \mod P = 0 \\ P_1\Pi^*(A_1) + P_2\Pi^*(A_2) & \text{else} \end{cases} \tag{5}$$

This implies that the minimum perimeter problem can be viewed as a tiling problem, because if P optimal shapes can be tiled in the grid so as to completely cover it (it is easy to see that in this case no overlap can occur) the resulting partition induced by these tiles is a provably optimal one. However, computational experience shows that even on the restricted class of rectangular grids, the lower bound in (5) is not always achievable. Consider for example the rectangular grid of dimensions $1 \times N$ with N even (shown in figure 6) to be equi-partitioned among 2 processors. The optimal partition has a total perimeter equal to $4(N/2+1)$ while the lower bound is $4\left\lceil 2\sqrt{N/2} \right\rceil$. Clearly, the relative distance defined by the ratio of the difference of the solution minus the one predicted from the lower bound over the one predicted grows as $\sqrt{N}$. This gap is due to the fact that the lower bound assumes domains large enough in both dimensions so as to fit the relatively square optimal shapes.

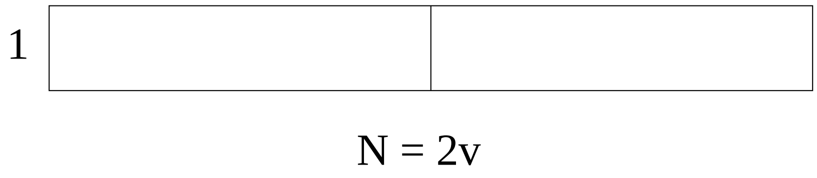

Figure 6: Optimal Partition for the MP($1 \times N$, 2)

In fact, the lower bound can fail to be attained even for square domains, as is the case for the MP(5×5,5), an optimal partition of which is shown in figure 7 (nevertheless, the theorems developed in the next section show that the lower bound (5) is good in an asymptotic sense). For the 5×5

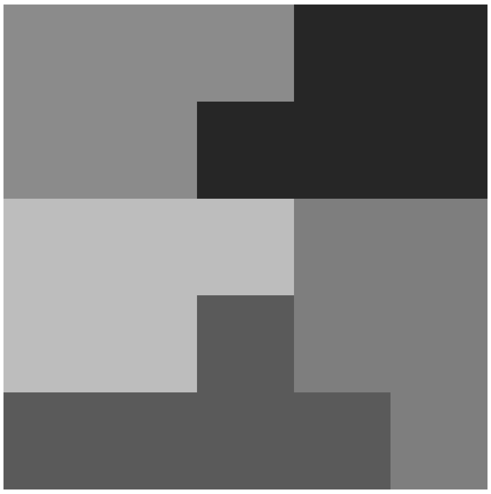

Figure 7: Optimal Partition for the MP(5×5, 5)

grid partitioned among 5 components, there is only one optimal shape (shown in figure 5) and there is no way to tile 5 such shapes in the 5×5 grid with no overlap. So the optimal perimeter is 52 instead of 50 which is the one predicted by the lower bound. One reason that the lower bound (5) fails to be tight is that the dimensions of the grid may not be large enough to accommodate the relatively square optimal shapes [CM96b]; in particular, we have the following lemma:

Lemma 2 *Assume that $M < N$ and that the following problem $(\mathcal{P})$ is feasible:*

$$\min_{h,w} \; h + w$$
$$s.t. \begin{cases} hw \geq A \\ h \leq M \\ w \leq N \\ h, w \in \mathbb{N}. \end{cases}$$

Let $(\mathcal{P}_{rel})$ denote the relaxed problem

$$\min_{h,w} \; h + w$$
$$s.t. \begin{cases} hw \geq A \\ h, w \in \mathbb{N}. \end{cases}$$

Assume that all optimal solutions of $(\mathcal{P}_{rel})$ violate at least one of the constraints of $(\mathcal{P})$. Then, an optimal solution of $(\mathcal{P})$ is $(h^*, w^*) = (M, \lceil \frac{A}{M} \rceil)$.

Proof: For each $h = 1 \ldots M$ that corresponds to a feasible point of $(\mathcal{P})$ (i.e. satisfies $\lceil \frac{A}{h} \rceil \leq N$), the w in the range $\lceil \frac{A}{h} \rceil \ldots N$ that yields the best objective is the value $\left\lceil \frac{A}{h} \right\rceil$. Thus, we need only find the number h in the range $1 \ldots M$ that corresponds to a feasible solution and minimizes the function $f(h) = h + \left\lceil \frac{A}{h} \right\rceil$. But the function $f(h)$ in the range $1 \ldots \left\lfloor \sqrt{A} \right\rfloor$ is non-increasing. To see this assume $i < \left\lfloor \sqrt{A} \right\rfloor$; we are going to show that $i + \left\lceil \frac{A}{i} \right\rceil \geq i + 1 + \left\lceil \frac{A}{i+1} \right\rceil$, or equivalently that

$$\left\lceil \frac{A}{i} \right\rceil - \left\lceil \frac{A}{i+1} \right\rceil \geq 1.$$

But the number $d := \frac{A}{i} - \frac{A}{i+1} = \frac{A}{i(i+1)} > 1$ as $i(i+1) < \left\lfloor \sqrt{A} \right\rfloor^2 \leq A$. Therefore, the ceilings of the two numbers $\frac{A}{i}$, $\frac{A}{i+1}$ are at a distance greater than, or equal to one, so $\left\lceil \frac{A}{i} \right\rceil - \left\lceil \frac{A}{i+1} \right\rceil \geq 1$.

As $M < \left\lfloor \sqrt{A} \right\rfloor$ (otherwise, there exists an optimal solution of $(\mathcal{P}_{rel})$ that does not violate the extra constraints of $(\mathcal{P})$ because it is shown in [YM92a] that there always exist an optimal solution of $(\mathcal{P}_{rel})$ that has $h^* = \left\lfloor \sqrt{A} \right\rfloor$; see also lemma 1.) an optimal solution of $(\mathcal{P})$ is $(h^*, w^*) = (M, \left\lceil \frac{A}{M} \right\rceil)$ and the optimal objective value of $(\mathcal{P})$ is $M + \left\lceil \frac{A}{M} \right\rceil$. ∎

The above lemma 2 confirms that when the domain is a sufficiently narrow horizontal band (M being small enough) so that no optimal shape from the collection of optimal shapes fits in the domain, then the optimal perimeter is $2(M + \left\lceil \frac{A}{M} \right\rceil)$.

2.3 Error Bounds for Equi-partitions of Rectangular Domains

In much of the analysis below, rather than dealing directly with perimeter, it is more convenient to use the concept of semi-perimeter (introduced earlier in the previous section). First, we consider the equi-partitioning problem MP($M \times N$, P) for rectangular grids of dimensions $M \times N$ and we assume perfect load balancing, i.e. that P divides MN and that each processor is therefore assigned $\mathcal{A}_p = \frac{MN}{P}$ cells.

A key observation is that for many instances of the problem, a stripe-decomposition of the domain is possible; that is, an optimal –or near optimal– partition exists, where the sub-domains form horizontal stripes of height approximately $\sqrt{\mathcal{A}_p}$ that partition the rows of the grid; observe the stripes of figure 8 which shows an optimal partition (i.e. one that matches the lower bound) of a 200×200 rectangular grid among 200 processors. To establish this claim, we are going to prove two lemmas. These two lemmas combined, guarantee the existence of such solutions for a large class of instances of the minimum perimeter problem (see also [CM96c]).

Lemma 3 *Given two nonnegative integers m, k there exist natural numbers a, b such that*

$$m = ak + b(k+1) \tag{6}$$

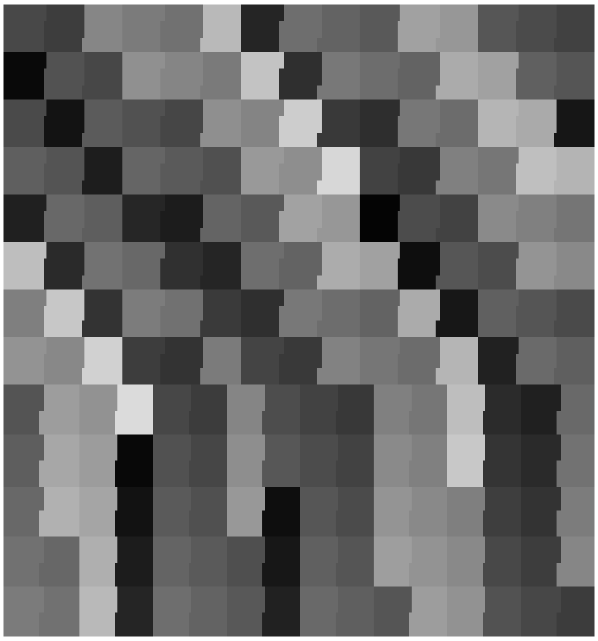

Figure 8: Optimal stripe partition of a 200x200 grid among 200 procs

iff $r_m = 0$ *or*

$$k \leq d_m + r_m$$

$$where\ d_m = m \div (k+1)$$

$$and\ r_m = m \mod (k+1).$$

Proof: The case $r_m = 0$ is trivial, so in the arguments below, we assume $r_m > 0$. From the definition of d_m and r_m we have

$$m = d_m(k+1) + r_m.$$

But this can be written as

$$m = (d_m - c)(k+1) + (r_m + c + kc)$$

for any c. Now, we can write m in the desired form (6) if k divides r_m+c and $0 \leq c \leq d_m$. The smallest c that satisfies this requirement is $k-r_m$, and thus, if $c = k-r_m \leq d_m$ then simply set $b = d_m - (k-r_m)$ $a = \frac{r_m+(k+1)(k-r_m)}{k} = k+1-r_m$. For the other direction, observe that $k-r_m$ is the smallest c that allows k to divide the number $r_m+(k+1)c$ so if this c is greater than d_m, there exists no natural such that the required decomposition is possible. ∎

A useful corollary of this lemma is the following:

Corollary 4 *Given two nonnegative integers m, k there exist natural numbers a, b such that equation (6) holds if $m \geq k(k-1)$.*

Proof: The corollary trivially holds for $k = 0$ or $k = 1$. Assume therefore $k \geq 2$. If $k(k-1) \leq m \leq k^2-1$, then $m = k^2 - r$ for some $r = 1 \ldots k$, and thus write $m = (r-1)k + (k-r)(k+1)$. Else $m \geq k^2$. For all m between k^2 and $k(k+1)-1$ we have $d_m = k-1$ and $r_m \geq 1$, so $k \leq d_m + r_m$ and the claim holds. For all m greater than or equal to $k(k+1)$ we have $d_m \geq k$ and so $k \leq d_m + r_m$ and the claim holds true again. ∎

The next lemma implies that the class of problems MP($N \times N$,N) is amenable to optimal-heights stripe decomposition. In other words, for all $N > 0$, we can partition the rows of the grid with a number of stripes, each of which has a height that is equal to the height of an optimal shape.

Lemma 5 *Given N, we can always find r near-rectangular optimal shapes*

(h_i, w_i, f_i) *–not necessarily distinct– such that*

$$\sum_{i=1}^{r} h_i = N.$$

Proof: We are going to show that we can always find two optimal shapes (h_1, w_1, f_1) and (h_2, w_2, f_2) where $f_1 < h_1$, $f_2 < h_2$, such that $ah_1 + bh_2 = N$ for some natural numbers a, b. Let $k = \left\lfloor \sqrt{N} \right\rfloor$.

- Assume $k(k+1) > N$. The discussion in section 2.2 implies that $(k, k, N - k^2)$ is an optimal shape and its semi-perimeter is $2k + 1$ (unless $N = k^2$ in which case the semi-perimeter is $2k$, and we can get a perfect partition using the optimal shape $(k, k, 0)$). Furthermore, trying the rectangle $(k + 1, k - 1)$ we get

  $$(k + 1)(k - 1) = k^2 - 1 < N$$

  and $f = N - k^2 + 1 < k + 1$ because $N < k(k + 1)$ so the shape $(k + 1, k - 1, N - k^2 + 1)$ is also an optimal shape. Both of these optimal shapes have fringe size less than the height of the corresponding block. Since $N \geq k^2$, by corollary 4 we can find two naturals a, b, such that $N = ak + b(k + 1)$.

- Next assume that $k(k+1) = N$. This simply means that the MP($N \times N$,N) has an optimal shape that is a rectangle and thus we can obtain a perfect partition using N rectangles of dimensions $k \times (k + 1)$ all oriented in the same way.

- Finally, assume $k(k + 1) < N$. Observe that $(k + 1)^2 > N$ from the definition of k. Now, the shapes $(k + 1, k, f)$ and $(k, k + 1, f)$ where

$f = N - k(k+1) < k+1$ are optimal. Again, because $N > k^2$ corollary 4 applies and the required decomposition of the rows of the grid is possible. Note that if $f = k$ the rectangle $k \times (k+2)$ is an optimal shape, and a perfect partition using N such rectangles all oriented the same way is possible.

∎

Lemma 5 implies that for the MP($N \times N$, N), it is possible to partition the rows of the grid into r stripes of heights that correspond to the height of optimal shapes for area N. Such a stripe will have area hN and can be filled with h shapes (not all of which need be optimal) of area N and height h using a *stripe-filling* process that fills the stripe with exactly h such shapes assigning them processor indices $1 \ldots h$. We present this routine in the form of pseudocode and then describe it in more detail, as this routine forms the basis of the arguments to be presented in the next theorem.

```
procedure stripeFill(h,A:integer; var str: grid)
/* input: h,A - the dimensions of the stripe (h x A)
   output: str - the processor index assignments of the cells
*/
begin
proc = 1;
area[proc] = 0;
for col = 1 to A
```

```
  for row = 1 to h
    str[row,col] = proc;
    area[proc] = area[proc] + 1
    if (area[proc] = A)
      proc = proc + 1;
      area[proc] = 0
    endif
  endfor
endfor
end;
```

In terms of placing optimal shapes, the effect of this process is to place the block-part of the current optimal shape so that its leftmost column occupies the first completely unassigned column of the stripe. If there exist any unassigned cells in the column to the left of this column, the routine places as much of the fringe as possible there. If there is some part of the fringe that does not fit there, the algorithm places this remainder at the top of the immediate right neighboring column of the block. However, if all fringe cells are assigned to the left and there remain neighboring cells on the left that are not assigned, the algorithm alters the shape by removing cells from the rightmost column of its block and using them to fill the residual left neighbors of the block (in this latter case, the perimeter of the resulting shape remains optimal).

In figure 9 we illustrate the placement of the first two shapes of a given

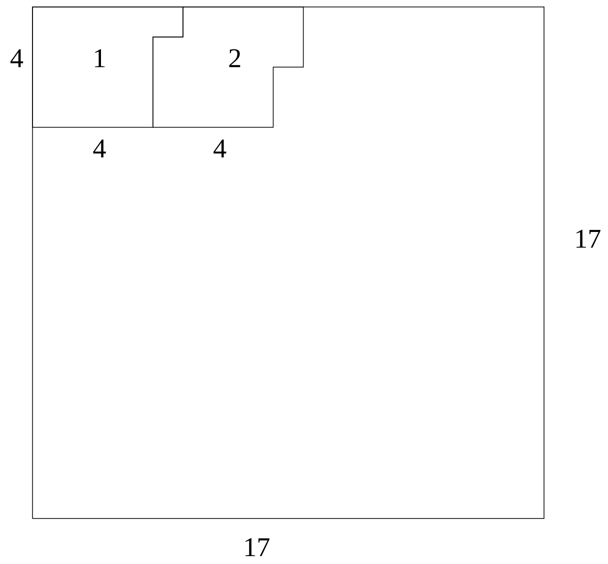

Figure 9: Placement of the initial two shapes for the MP(17×17, 17)

input string for the MP(17×17,17) problem within a stripe. Note that each of these shapes is a 4×4 rectangle accompanied by a fringe cell. The second shape has been modified by the stripe-filling process, but its total perimeter is still optimal (equal to 18).

Using lemma 3 and 5 and corollary 4 together with the `stripeFill(...)` routine we can proceed to show that a large class of minimum perimeter problems is amenable to a decomposition that yields partitions that become asymptotically optimal as the area that each processor is assigned grows to infinity. This class of problems is the equi-partitioning problems MP($M \times N$, P) with $P \geq \max(M, N)$. We prove this fact in a series of constructive theorems starting with the case $M = N = P$. In the following, the relative distance of a partition of a grid $\mathcal{G}$ among P processors from the lower bound (5) is defined to be $\frac{z-L(\mathcal{A}_p,P)}{L(\mathcal{A}_p,P)}$ where z is the perimeter of the partition.

Theorem 6 *The MP($N \times N$, N) problem (equi-partition an $N \times N$ grid among N components) has a solution whose relative distance δ from the lower bound (5) satisfies*

$$\delta < \frac{1}{\lceil 2\sqrt{N} \rceil}. \tag{7}$$

Proof: Let (h_i, w_i, f_i) denote an optimal shape of height h_i, width w_i and fringe size f_i for area size $\mathcal{A}_p = \frac{N \times N}{N} = N$ where $f_i < h_i$.

As we have already shown in lemma 5, we can always find r shapes –not necessarily distinct– such that $\sum_{i=1}^{r} h_i = N$. These numbers $h_1, \ldots, h_r$

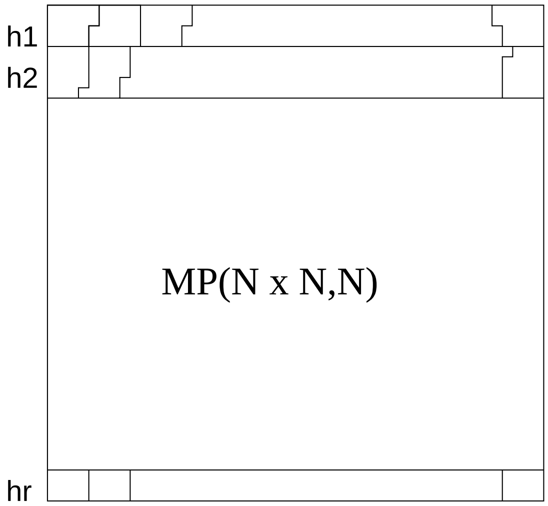

Figure 10: Stripe Form of the Partition

induce a partition on the rows of the grid (see figure 10). The first h_1 rows of the grid are called the first stripe, the following h_2 rows are called the second stripe etc.

Now, the i-th stripe, can be filled with h_i components using the shape (h_i, w_i, f_i). In order to do this simply use the `stripeFill(...)` routine described earlier. In this manner, the stripe is filled using exactly h_i components, because the area of the stripe is $h_i N$, and the total area of h_i components is $h_i N$ also.

If $f_i = 0$ then no error occurs in the i-th stripe. If $f_i > 0$, the error in this stripe can be no more than $f_i - 1$. To see this observe that each component is either optimal (if it occupies $w_i + 1$ columns of the stripe) or its semi-perimeter is suboptimal by 1 (if the fringe part of the shape is split between the immediate left and right columns of the block). So, we can measure the distance from lower bound in the stripe by counting the number of the suboptimal shapes, or equivalently, by counting the "surplus" columns corresponding to regions that occupy $w_i + 2$ columns.

In the stripe, assume there are e_0 shapes that fill completely $w_i - 1$ columns of the stripe, and occupy part of their immediate left and right neighboring columns, e_0^+ shapes that fill w_i columns of the stripe and occupy part of one immediate neighboring column, and e_i shapes that are suboptimal , that is they fill w_i columns of the stripe, and they occupy part of both their immediate neighboring columns. Thus, letting t denote the number of columns containing more than one component index, we have

$$e_0 + e_0^+ + e_i = h_i \tag{8}$$

$$h_i w_i + f_i = N \tag{9}$$

$$e_0(w_i - 1) + e_0^+ w_i + e_i w_i + t = N \tag{10}$$

from which, after substitution, we conclude that

$$t = f_i + e_0.$$

Let us now associate each of the t doubly indexed columns with the component corresponding to the block to its left. Then the shapes corresponding to e_0 each contribute to t as do the e_i suboptimal shapes and the first e_0^+ shape at the left end of the stripe. Therefore, $e_0 + 1 + e_i \leq t$. Combining this with the preceding equation implies

$$e_i \leq f_i - 1.$$

Therefore, the semi-perimeter error in each stripe is not more than $f_i - 1$ and the stripes cover the grid completely and with no overlap using a total of $\sum_{i=1}^{r} h_i = N$ components, so the relative error is bounded by

$$\delta \leq \frac{1}{N\left\lceil 2\sqrt{N}\right\rceil} \sum_{i=1}^{r} (f_i - 1).$$

Defining $\pi_i \equiv \frac{f_i}{h_i}, \pi = \max_i \pi_i$, we have $\pi_i \in [0, 1),\ \pi \in [0, 1)$ so substituting in the above we get

$$\delta \leq \frac{1}{N\left\lceil 2\sqrt{N}\right\rceil} \sum_{i=1}^{r} (f_i - 1) = \frac{1}{N\left\lceil 2\sqrt{N}\right\rceil} \sum_{i=1}^{r} (\pi_i h_i - 1) \leq \frac{\pi N - r}{N\left\lceil 2\sqrt{N}\right\rceil} < \frac{1}{\left\lceil 2\sqrt{N}\right\rceil}$$

as stated. ∎

It is worth mentioning that in the case when the fringe f_i of an optimal shape (h_i, w_i, f_i) divides its height h_i exactly, then there can be no surplus columns and therefore the error in a stripe using this shape is zero. The same zero error behavior of stripes occurs when $f_i \leq 1$. This implies that it is not unlikely in the best near-optimal solutions to observe a large number of stripes of zero total error (more details and experimental results verifying this claim can be found in [Mar96]).

The techniques used in the proof of theorem 6 can be used to prove that similar quality solutions exist when the number of rows equals the number of processors and is greater than or equal to the number of columns of the grid. However, it now may become necessary to consider stripes of sub-optimal height (but by no more than 1).

Theorem 7 *The MP($M \times N$,M) with $M \geq N$ has a solution whose total perimeter possesses a relative distance δ from the lower bound that satisfies*

$$\delta < \frac{1}{\lceil 2\sqrt{N} \rceil}. \tag{11}$$

Proof: The proof is very similar to the proof of theorem 6. We are going to partition the M rows of the grid into $r_1 + r_2$ stripes having lengths $h_1, \ldots, h_{r_1}$, $h^s_1, \ldots, h^s_{r_2}$. The first r_1 stripes will have optimal heights h_i while the last r_2 stripes will have a sub-optimal height h^s_i. The only case in which we use these heights is when N is a perfect square $N = k^2$. In this case, the stripe-filling process is based on sub-optimal shape $(k+1, k-1, 1)$ which has an area of $(k+1)(k-1)+1 = N$ and a semi-perimeter equal to $2k+1$. A non-optimal height is used to ensure that a partition of M analogous to lemma 5 is possible.

Let $k = \left\lfloor \sqrt{N} \right\rfloor$.

- Assume first $k(k+1) > N$. Furthermore, assume $N \neq k^2$. Under these assumptions, the shapes $(k, k, N-k^2)$ and $(k+1, k-1, N-k^2+1)$ can be used to partition the grid. Applying the technique described in the previous theorem, in the i-th stripe we can place h_i shapes, and the error in each of them is

$$e_i \leq f_i - 1.$$

It only remains to prove that we can find nonnegative integers a, b such that

$$M = ak + b(k+1). \tag{12}$$

But since $M \geq N \geq k^2$, from corollary 4, we have that equation (12) holds.

Now, assume that $N = k^2$. In this case, the shape $(k+1, k-1, 1)$ is sub-optimal by 1 as its semi-perimeter is $2k+1$. Nevertheless, the area size of this shape is N, and we can use $k+1$ shapes to fill a stripe of height $h_i^s = k+1$. The absolute error in such a stripe will be

$$e_i = h_i^s = k+1.$$

where the term h_i^s comes from the fact that each shape used in this stripe has a semi-perimeter that is suboptimal by one. Note that since $f_i^s = 1$, there exist no surplus columns in such a stripe. From the discussion above, we have that $M = ak + b(k+1)$ for some $a, b \in \mathbb{N}$, so we can partition the rows of the grid as desired. Now setting $r_1 = a$ and $r_2 = b$ we bound the total relative distance from above by

$$\delta \leq \frac{1}{M\left\lceil 2\sqrt{N}\right\rceil} \sum_{i=1}^{r_2} h_i^s$$

and since

$$\sum_{i=1}^{r_2} h_i^s = M - r_1 k$$

we get

$$\delta \leq \frac{1}{M\left\lceil 2\sqrt{N}\right\rceil} \left[M - \left\lfloor \sqrt{N}\right\rfloor r_1\right].$$

- Next, assume that $N = k(k+1)$. This means that $(k+1, k, 0)$ and $(k, k+1, 0)$ are optimal rectangles. Since $M \geq N \geq k^2$ by corollary 4 we can always write $M = ak + b(k+1)$ for some $a, b \in \mathbb{N}$. Note, that the error in each stripe is zero, which results in a perfect partition.

- Finally, in the case $N > k(k+1)$, the shapes $(k+1, k, f)$ and $(k, k+1, f)$ are optimal shapes for the MP($M \times N$,M). Note that $f = N - k(k+1)$, and if $f = k$ then the shape $(k, k+1, k)$ is really the optimal rectangle $(k, k+2, 0)$. Using the same arguments again, we can partition the rows of the grid by finding $a, b \in \mathbb{N}$ such that $M = ak + b(k+1)$. The error in the i-th stripe will be

$$e_i \leq f_i - 1.$$

So we have shown that in all cases there exists a solution whose total perimeter has a relative distance from the theoretical lower bound that is

$$\delta \leq \frac{\pi M - r}{M \left\lceil 2\sqrt{N} \right\rceil}$$

for some $\pi \in [0, 1)$ and $r \in \mathbb{N}$. ∎

The theorem that we just proved will be used as an argument for the proof of the more general theorems proved in the following. The general idea is to reduce whatever grid is given into a series of rectangular grids of dimensions $M \times \mathcal{A}_p$ to be partitioned among M processors (with $M \geq \mathcal{A}_p$). Then if the remainder of the original grid is a small enough area compared to the whole of the grid, the relative distance of the resulting partition cannot grow too much if one partitions the remainder in a careful way. So, next we present the final and most general theorem for equi-partitioning a rectangular grid in the case that the number of processors divides the total area of the grid; the asymptotically optimal behavior of the class of grids to be presented was first established in [CM96c]. The following result [CM96b] is an improvement on the error bound, and the proof is more pleasing aesthetically.

Theorem 8 *Assuming P divides MN and that $P \geq max(M, N)$ the minimum perimeter problem MP($M \times N$,P) has a feasible solution whose relative distance*

δ from the lower bound satisfies

$$\delta < \frac{1}{\sqrt{\mathcal{A}_p}} + \frac{1}{\mathcal{A}_p}. \tag{13}$$

Thus the error bound δ converges to zero as $\mathcal{A}_p$ (the area of each processor) tends to infinity.

Proof: The grid is shown in figure 11. Note that $\mathcal{A}_p \leq \min\{M, N\}$

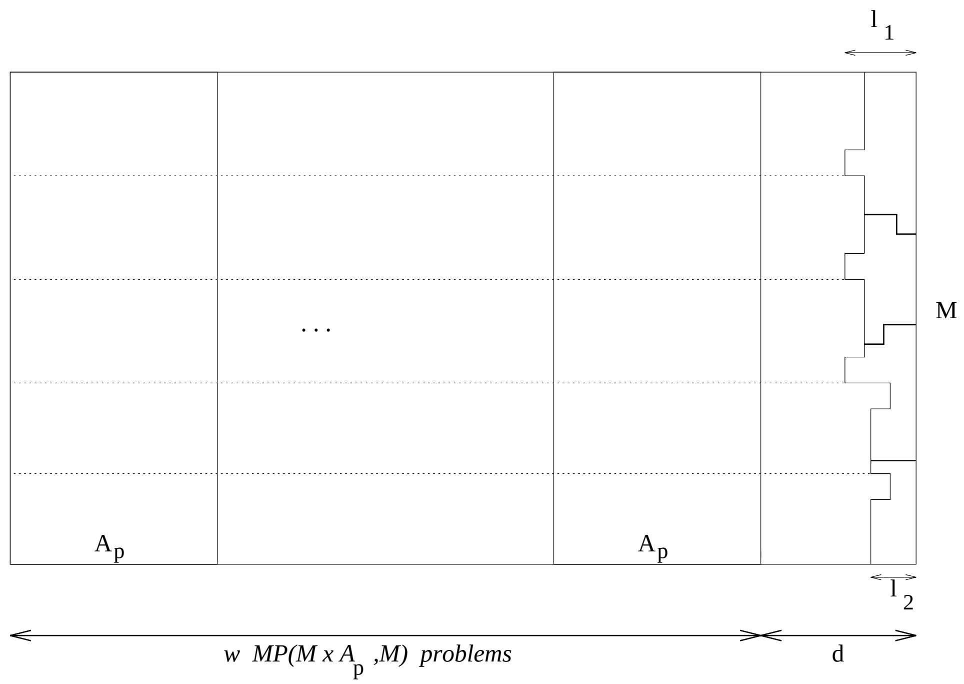

Figure 11: MP($M \times N$,P), $P \geq \max(M, N)$

and write $N = w\mathcal{A}_p + d$ for some naturals $w \geq 1$ and $d < \mathcal{A}_p$. Define $k = \left\lfloor \sqrt{\mathcal{A}_p} \right\rfloor$. Observe that the problem can be decomposed into w MP($M \times \mathcal{A}_p$,M) problems, and a MP($M \times d$,$Md/\mathcal{A}_p$). In each of the w problems MP($M \times \mathcal{A}_p$,M), use the stripe decomposition method of theorem 7 to get a total absolute perimeter error $e < 2wM$. This striping technique (which partitions the rows of the grid into $r \leq M/k$ stripes of height $h_1, \ldots, h_r$) is continued over the last d columns in each stripe until no additional shape of area $\mathcal{A}_p$ can be placed in the stripe. Let p denote the number of processors that have not been assigned. The stripe decomposition in the last d columns thus placed $\frac{Md}{\mathcal{A}_p} - p$ processors, each of which may have an error in perimeter of no more than two.

The stripe decomposition for MP($M \times \mathcal{A}_p$,M) uses at most two different heights. Arrange the stripes of the grid so that all stripes that use the first height are used in the top rows of the grid which we will refer to as area 1, and all the stripes that use the second shape are in the (remaining) bottom rows which we will refer to as area 2. Let l_i $i = 1, 2$ denote the maximum number of columns in area i that contain unassigned grid cells, and without any loss of generality, assume that $l_1 \geq l_2 \geq 0$.

We place the last p processors in the remaining area using the following "orthogonal stripe filling" algorithm that approximates the optimal shapes established in lemma 2: starting from the top row of the grid, keep assigning the unassigned cells row-wise (interchanging left to right and then right to left) until the processor has $\mathcal{A}_p$ cells.

To compute the error bound in perimeter of the last p processors that were placed in the grid using this "orthogonal stripe filling" algorithm we compute the length of the boundary enclosing the region they occupy, plus the length of the border between processors, then subtract the lower bound. Thus, the maximum error in this region is

$$e < (2M + l_1 + l_2) + (2r - 1 + l_1 - l_2) + 2\left[(p-1)l_1 + (p-1)\right] - 2(p\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil).$$

The first six terms in the RHS of the inequality account for the left, right, top and bottom borders of this region, the next two terms account for the inner borders, and the last term is the lower bound. Note that the perimeter of the left border includes four terms $(M+(2r-1)+l_1-l_2)$ because it is not a straight line. Thus the total relative distance of the perimeter of the solution from the lower bound satisfies:

$$\begin{aligned}\delta &< \frac{2\left[wM + \frac{Md}{\mathcal{A}_p} - p + (p-1)l_1 + p - 1 - p\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil\right]}{2M\frac{w\mathcal{A}_p+d}{\mathcal{A}_p}\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil} \\ &\quad + \frac{2M + l_1 + l_2 + (2r-1) + l_1 - l_2}{2M\frac{w\mathcal{A}_p+d}{\mathcal{A}_p}\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil}\end{aligned}$$

or

$$\delta < \frac{wM + \frac{Md}{\mathcal{A}_p} + (p-1)l_1 - 1 - p\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil + M + l_1 + r}{M\frac{w\mathcal{A}_p+d}{\mathcal{A}_p}\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil}.$$

But for all $\mathcal{A}_p \geq 2$, we have $l_1 \leq \left\lfloor\sqrt{\mathcal{A}_p}\right\rfloor + 2 \leq \left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil$ which implies that $(p-1)l_1 + l_1 = pl_1 \leq p\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil$, and since $r \leq \frac{M}{\left\lfloor\sqrt{\mathcal{A}_p}\right\rfloor}$ we get

$$\begin{aligned}\delta &< \frac{M(w\mathcal{A}_p + d) + M\mathcal{A}_p + \frac{M\mathcal{A}_p}{\left\lfloor\sqrt{\mathcal{A}_p}\right\rfloor}}{M(w\mathcal{A}_p + d)\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil} \\ &= \frac{1}{\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil} + \frac{\mathcal{A}_p}{(w\mathcal{A}_p + d)\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil} + \frac{\mathcal{A}_p}{(w\mathcal{A}_p + d)\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil\left\lfloor\sqrt{\mathcal{A}_p}\right\rfloor}.\end{aligned}$$

It's easy to show that $\forall x \geq 1 \;\; x^2 \leq \lceil 2x\rceil\lfloor x\rfloor$ so (since $\mathcal{A}_p \leq M$, $\mathcal{A}_p \leq N$)

$$\delta < \frac{2}{\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil} + \frac{1}{N} < \frac{1}{\sqrt{\mathcal{A}_p}} + \frac{1}{\mathcal{A}_p}.$$

∎

It is easy to use the ideas behind the proof of theorem 8 to establish a similar result in the case where P does not divide MN.

Theorem 9 *Assume P does not divide MN and that $P \geq max(M,N)$; the minimum perimeter problem MP($M \times N$,P) has a feasible solution whose relative distance δ from the lower bound satisfies*

$$\delta < \frac{1}{\sqrt{A_1}} + \frac{1}{\sqrt{A_2}} + \frac{1}{A_1} \tag{14}$$

where $A_1 = \lfloor MN/P \rfloor$ and $A_2 = \lceil MN/P \rceil$. Thus the error bound δ converges to zero as A_1, A_2 (the areas of the processors) tend to infinity.

Proof: Decompose the grid among the P processors using the stripe decomposition and orthogonal stripe filling techniques discussed in the proof of theorem 8, initially assigning an area $A_1 = MN \div P$ to *all* P processors. Note that N can be written as $wA_1 + d$ for some naturals $w > 0$ and $0 \leq d < A_1$. The w MP($M \times A_1$,M) problems cover the first wA_1 columns of the grid. The stripe filling is continued over the last d columns in the same manner as in the previous theorem. Orthogonal stripe filling is used for the remaining processors (if there are any). This leaves $P_2 = MN \mod P$ grid cells unassigned near the bottom right corner of the grid (the gray area in figure 12). Assign each of these cells to the last P_2 processors; the perimeter of these cells is at most $4P_2$. Recall that the lower bound in perimeter is a non-decreasing function of the area of each processor, and therefore, the absolute perimeter error of the

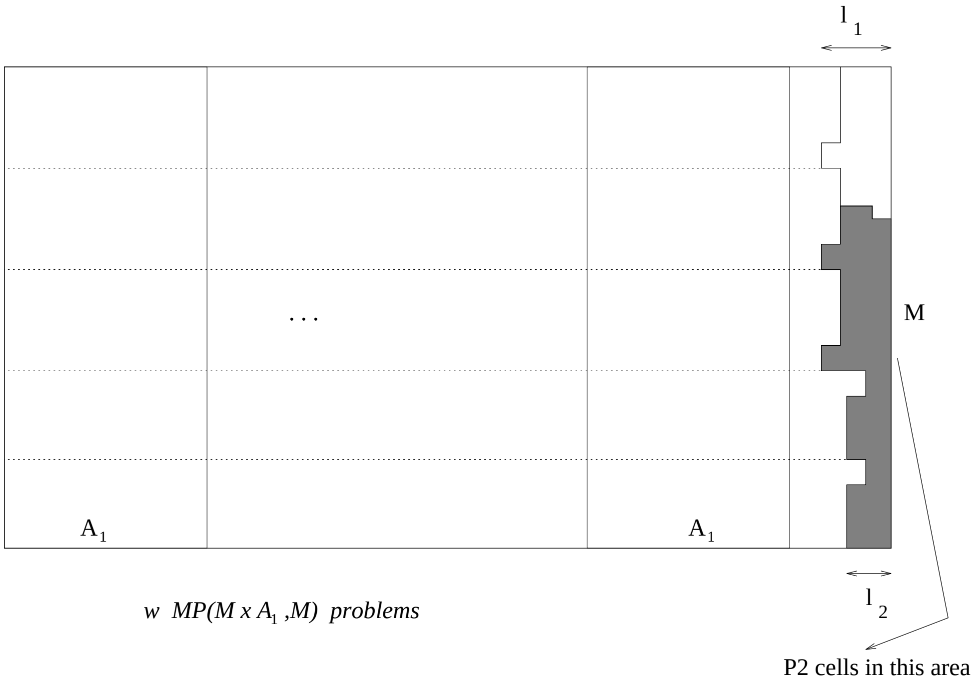

Figure 12: MP($M \times N$,P), $P \geq \max(M, N)$, $MN \mod P \neq 0$

last P_2 processors (having area $A_2 = A_1 + 1$) can be no larger than the absolute error computed in the stripe-decomposition of their first A_1 cells plus 4 (for the extra cell). Thus, the relative distance of the perimeter of the partition from the lower bound is

$$\delta < \frac{1}{\sqrt{A_1}} + \frac{1}{A_1} + \frac{4P_2}{2P_2\left\lceil 2\sqrt{A_2}\right\rceil}$$

(since the lower bound is $2(P_1\left\lceil 2\sqrt{A_1}\right\rceil + P_2\left\lceil 2\sqrt{A_2}\right\rceil) \geq 2P_2\left\lceil 2\sqrt{A_2}\right\rceil$) and therefore

$$\delta < \frac{1}{\sqrt{A_1}} + \frac{1}{\sqrt{A_2}} + \frac{1}{A_1}.$$

∎

Theorems 8 and 9 guarantee the existence of good quality solutions for the MP($M \times N$,P) as long as the number of processors dominates the dimensions of the grid. These solutions become optimal in an asymptotic sense as the area that each processor is assigned to grows to infinity. The drawback of requiring too many processors in order to guarantee good partitioning schemes is real; parallel computing in networks of workstations has attracted a lot of attention because it has been shown to be a viable and much cheaper alternative to massively parallel supercomputers. In such an environment however, the number of available processors may not be as large as the domain (which is assumed big enough to require the use of parallel processing for the efficient solution of the problem in hand). Still, the computational results show that even when the number of processors is much less than the dimensions of the grid, stripe decomposition is able to find very good quality partitions. Furthermore, the technique used to fill the last d columns of the grid in theorem 8, can be used to show that in the case where $N \leq P < M$ there exists a partition whose total perimeter approaches the lower bound (asymptotically) if $\frac{M}{PN}$ tends to zero.

Theorem 10 *If M, N and P satisfy $N \leq P < M$ and P divides MN, then the minimum perimeter problem MP($M \times N$,P) has a feasible solution whose relative distance δ from the lower bound satisfies*

$$\delta < \frac{1}{\left\lceil 2\sqrt{\mathcal{A}_p} \right\rceil} + \frac{1}{N} + \frac{\mathcal{A}_p}{N \left\lceil 2\sqrt{\mathcal{A}_p} \right\rceil}$$

where $\mathcal{A}_p = MN/P$ is the area of each processor.

Proof: Let k (as before) be $\left\lfloor\sqrt{\mathcal{A}_p}\right\rfloor$. From the hypothesis, $P \geq N$, and since $\mathcal{A}_p = \frac{MN}{P}$ we get $M \geq \mathcal{A}_p \geq k^2 \geq k(k-1)$, so by corollary 4, M can be written as $ak + b(k+1)$. This means that the grid can be decomposed exactly like the grid of figure 11 except for the first $w\mathcal{A}_p$ columns which do not exist. Following exactly the same arguments for computing the perimeter of the solution in the last d columns of figure 11 then, gives us

$$\delta < \frac{2\left[(P-p)+(p-1)l_1+(p-1)\right]}{2\frac{MN}{\mathcal{A}_p}\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil} + \frac{(2M+l_1+l_2)+(2r-1)+(l_1-l_2)-2p\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil}{2\frac{MN}{\mathcal{A}_p}\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil}$$

from which, after substitution ($l_1 \leq \left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil$ and $r \leq \frac{M}{\left\lfloor\sqrt{\mathcal{A}_p}\right\rfloor}$) we get

$$\delta < \frac{1}{\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil} + \frac{1}{N} + \frac{\mathcal{A}_p}{N\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil}.$$

∎

It is easy to check that if M, N and P grow large in such a manner so that $\frac{M}{PN} \to 0$ then $\delta \to 0$ too. For example, by letting $P = N = \mathcal{A}_p^{\frac{1}{2}+\epsilon}$, and $M = \mathcal{A}_p$, then as $\mathcal{A}_p$ tends to infinity, $\delta \to 0$.

Similarly, when P does not divide the total area $A = MN$ of the grid, we have

Theorem 11 *If M, N and P satisfy $N \leq P < M$ and P does not divide MN, then the minimum perimeter problem MP($M \times N$,P) has a feasible solution* *whose relative distance δ from the lower bound satisfies* $\delta < \frac{1}{\lceil 2\sqrt{A_1}\rceil} + \frac{1}{N} + \frac{A_1}{N\lceil 2\sqrt{A_1}\rceil} + \frac{1}{\sqrt{A_2}}$ *where* $A_1 = \lfloor MN/P \rfloor$ *and* $A_2 = \lceil MN/P \rceil$.

Proof: Since P does not divide MN, we have $A_1 = \left\lfloor \frac{MN}{P} \right\rfloor$ and because $\frac{N}{P} \in (0,1)$ we have $M > A_1 \geq k^2 > k(k-1)$ and thus the grid can be decomposed as shown in figure 12, except the first wA_1 columns which do not exist. Using the stripe decomposition method as described in theorem 9, we initially assign an area $A_1 = MN \div P$ to *all* processors, and fill the last bottom-right P_2 unassigned cells with the remaining cell of each of the P_2 processors. The error in perimeter of these last P_2 processors can be no larger than the absolute error computed in the stripe decomposition of their first A_1 cells plus four (for the extra cell). Using theorem 10, the relative distance of the perimeter of the partition from the lower bound is less than $\frac{1}{\lceil 2\sqrt{A_1}\rceil} + \frac{1}{N} + \frac{A_1}{N\lceil 2\sqrt{A_1}\rceil} + \frac{4P_2}{2P_2\lceil 2\sqrt{A_2}\rceil}$ (since the lower bound on perimeter is greater than $2P_2\left\lceil 2\sqrt{A_2}\right\rceil$). ∎

2.4 A Knapsack Approach to Stripe Decomposition

In the case when the grid is a rectangle of dimensions $M \times N$ and P divides the total area MN, partitioning the rows of the grid among a set of stripe heights $h_1, \ldots, h_n$ gives rise to an easily computable distance from the lower bound associated with the p_i processors that will be assigned to the i-th stripe by the stripe-filling process. This distance is computed in [Mar96] to be

$$e_i = 2\left[N + p_i(h_i+1) - \frac{p_i}{r_i} - p_i\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil\right]$$

where r_i is the so-called repeat count of the patterns of shapes occuring in a stripe and is equal to

$$r_i = \min\{s_t \mid s_t = \frac{th_i}{f_i}, s_t \in \mathbb{N}\ ,\ t = 1, 2, \dots\}$$

. Notice that the numbers e_i and r_i are valid for non-optimal stripe heights as well, as long as the stripe height h_i is such so that an integer number of processors can cover the area of the stripe (that is equal to h_iN) exactly, i.e. as long as $\mathcal{A}_p$ divides h_iN.

So a natural question to ask is which set of stripes minimizes the total error bound. The problem (which we denote as (K)) is then a knapsack problem, and can be formulated as follows:

$$\min \sum_{i=1}^{n} e_i x_i \tag{15}$$

$$s.t. \begin{cases} \sum_{i=1}^{n} h_i x_i = M & \\ x_i \in \mathbb{N} & i = 1 \dots n \end{cases}$$

The knapsack constraint simply requires the selected stripes to cover exactly the rows of the grid, and the objective function is the distance of the perimeter of the resulting partition from the lower bound. Under the assumption that P divides MN, it is possible to prove that these knapsack constraints are feasible.

Note that P is not required to dominate the individual dimensions of the grid as in theorem 8. But there is no guarantee that the solution of this minimization problem will differ only slightly from the lower bound either. However, when combined, the two theorems suggest a very powerful method for equi-partitioning large rectangular grids, namely applying a knapsack procedure to determine the best stripe decomposition. Computational results show that optimal and near-optimal solutions are obtained very rapidly by using the partitions produced by MSP, an algorithm that solves the knapsack problem described above via a routine described in [MT90], and then using these stripes as input to the stripe filling process for cell assignment as described in [CM96c].

2.5 Error Bounds for Equi-partitions of Irregular-boundary Domains

We now consider a procedure for equi-partitions of *any* uniform 5-point grid among P processors using stripe decomposition if the grid contains a relatively large rectangular area. The partition obtained will differ only slightly from the lower bound (5).

Theorem 12 *Let $\mathcal{G}$ denote a 5-point uniform grid with total area $\mathcal{A}(\mathcal{G})$, to be equi-partitioned among P processors, each having an area $\mathcal{A}_p = \frac{\mathcal{A}(\mathcal{G})}{P}$. Let (M_0, N_0) denote the dimensions of the largest rectangle that can fit in $\mathcal{G}$, and let P_0 denote the maximum number of processors that can fit in this rectangle.*

Then, if $P_0 \geq \max(M_0, N_0)$ *the MP(*$\mathcal{G}, P$*) possesses a solution whose relative distance* δ_G *from the lower bound (5) satisfies* $\delta_G < \frac{1}{\mathcal{A}_p} + \frac{1}{\sqrt{\mathcal{A}_p}} + \left[1 - \frac{P_0 \mathcal{A}_p}{\mathcal{A}(\mathcal{G})}\right] \sqrt{\mathcal{A}_p}$.

Proof: From theorem 8, any rectangular grid of dimensions $M_0 \times N_0$

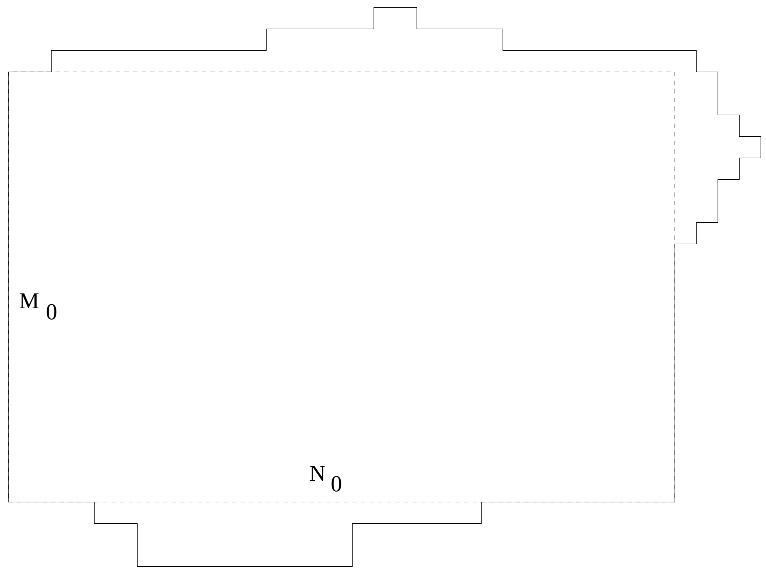

Figure 13: The Embedded Rectangle

can be partitioned among $P \geq \max(M_0, N_0)$ processors (using stripe decomposition) in such a way so that the relative distance of the perimeter of the solution from the lower bound is less than the RHS of (13), where $\mathcal{A}_p$ is simply the area of each processor. Now, consider a general grid $\mathcal{G}$ (see for example figure 13); the number of processors that can be assigned to the largest rectangle $M_0 \times N_0$ that can fit in this grid is $P_0 = \left\lfloor \frac{M_0 N_0}{\mathcal{A}_p} \right\rfloor$ and this number is by hypothesis greater than or equal to $\max(M_0, N_0)$. Using stripe decomposition, it is possible therefore to assign P_0 processors in this rectangular area with a total relative error

δ_0 in perimeter that is less than the expression in the RHS of (13). The rest of the grid (as well as possibly the right bottom part of the rectangle that might have been left unassigned using the P_0 processors) will be assigned using the remaining $P - P_0$ processors. In the worst possible case, each of the unassigned cells will have a perimeter of 4 and there are

$$\mathcal{A}_0^c(\mathcal{G}) = \mathcal{A}(\mathcal{G}) - P_0\mathcal{A}_p$$

cells left unassigned. So the total relative error of the solution from the lower bound is bounded from above by

$$\begin{aligned}
\delta_G <& \delta_0 + \frac{4\mathcal{A}_0^c(\mathcal{G}) - 2(P-P_0)\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil}{2P\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil} \\
<& \delta_0 + \frac{(P-P_0)\mathcal{A}_p}{P\sqrt{\mathcal{A}_p}} \\
<& \frac{1}{\mathcal{A}_p} + \frac{1}{\sqrt{\mathcal{A}_p}} + \\
&+ \left[1 - \frac{P_0\mathcal{A}_p}{\mathcal{A}(\mathcal{G})}\right]\sqrt{\mathcal{A}_p}
\end{aligned}$$

which ends the proof. ∎

As an easy corollary we obtain the following:

Corollary 13 *Let $\mathcal{G}^k$ be a sequence of 5-point uniform grids. Assume that $\mathcal{G}^k$ contains a rectangle of dimensions (M_0^k, N_0^k), has total area $\mathcal{A}(\mathcal{G}^k)$, and which is to be equi-partitioned among P^k processors each having an area $\mathcal{A}_p^k$. If $P_0^k := \left\lfloor \frac{M_0^k N_0^k}{\mathcal{A}_p^k} \right\rfloor \geq \max(M_0^k, N_0^k)$, then if $\left[1 - \frac{P_0^k \mathcal{A}_p^k}{\mathcal{A}(\mathcal{G}^k)}\right]\sqrt{\mathcal{A}_p^k} \to 0$ as $\mathcal{A}_p^k \to \infty$ the sequence MP($\mathcal{G}^k, P^k$) has solutions with relative distances from the lower bound $\delta_G^k \to 0$.*

The above results show that asymptotically, grids that are near-rectangular can be optimally partitioned using stripe decomposition. A class of such grids that satisfies the conditions of corollary 13 are elongated trapezoids to be partitioned among a large number of processors (such grids arise from simulations in chemical engineering). Consider for example the trapezoid shown in figure 14; the dimensions of the main rectangle of this trapezoid are $M \times N$. Let $M = N^2/k \geq N$ where k is an integer less than or equal to N. It is desired to equi-partition this grid among $P = 2M$ processors. The total area

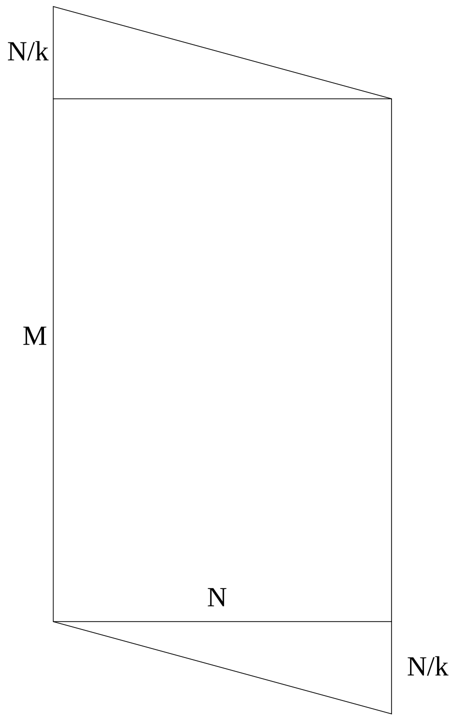

Figure 14: Elongated Trapezoid Example

of the trapezoid is $\mathcal{A}(\mathcal{G}) = MN + \frac{N^2}{k} = \frac{N^3+N^2}{k}$, and the area of each processor is $\mathcal{A}_p = \frac{\mathcal{A}(\mathcal{G})}{P} = \frac{1}{2}(N+1)$. The number of processors that can be placed in the $M \times N$ rectangle is $P_0 = \lfloor MN/\mathcal{A}_p \rfloor \geq \frac{MN}{\mathcal{A}_p} - 1 = \frac{2N^3 - k(N+1)}{k(N+1)}$ which is greater than or equal to both of M and N for all $N \geq 3$. Furthermore, we have that

$$\begin{aligned}\left[1 - \frac{P_0 \mathcal{A}_p}{\mathcal{A}(\mathcal{G})}\right] \sqrt{\mathcal{A}_p} &= \left[\frac{P - P_0}{P}\right] \sqrt{\mathcal{A}_p} \\ &\leq \left[\frac{1}{N+1} + \frac{k}{2N^2}\right] \sqrt{\mathcal{A}_p} \\ &\leq \frac{1}{\sqrt{N+1}} + \frac{\sqrt{N+1}}{N}\end{aligned}$$

and this last expression tends to zero as $N \to \infty$. Therefore such trapezoids can be partitioned efficiently in such a way so that as N becomes larger, the partitions' perimeter approach the lower bound.

It is possible to extend the ideas behind the previous theorem, to include 5-point uniform grids that contain a finite number of relatively large disjoint rectangular areas (see figure 15).

Theorem 14 *Let $\mathcal{G}$ denote a 5-point uniform grid with total area $\mathcal{A}(\mathcal{G})$, to be equi-partitioned among P processors, each having an area $\mathcal{A}_p = \frac{\mathcal{A}(\mathcal{G})}{P}$. For a finite n and $i = 1 \ldots n$ let (M_i, N_i) denote the dimensions of the n largest disjoint rectangles whose union fits in $\mathcal{G}$, and let P_i denote the maximum number of processors that can fit in each of these rectangles. Then, if $P_i \geq \max(M_i, N_i)$ the MP($\mathcal{G}, P$) possesses a solution whose relative distance δ_G from the lower bound (5) satisfies*

$$\delta_G < n \left[\frac{1}{\mathcal{A}_p} + \frac{1}{\sqrt{\mathcal{A}_p}}\right] + \left[1 - \frac{\sum_{i=1}^{n} P_i \mathcal{A}_p}{\mathcal{A}(\mathcal{G})}\right] \sqrt{\mathcal{A}_p}.$$

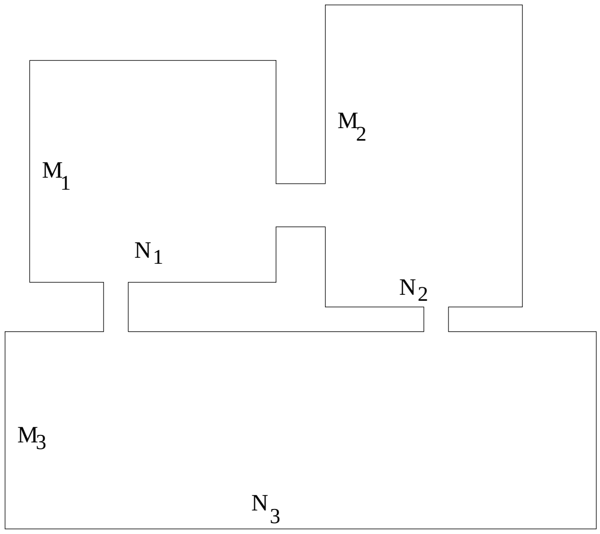

Figure 15: Many Embedded Disjoint Rectangles

Proof: In each of the disjoint rectangles $M_i \times N_i$, applying stripe decomposition will place all P_i processors with a relative error that will be less than $\frac{1}{\mathcal{A}_p} + \frac{1}{\sqrt{\mathcal{A}_p}}$. In the rest of the grid (including the possible bottom-right area of each of these rectangles that might have been left unassigned), in the worst case, when each cell is assigned (using a scanning procedure that scans all of the grid cells and assigns each unassigned grid cell with a processor index that corresponds to any processor that hasn't been assigned $\mathcal{A}_p$ cells yet) it will get a perimeter of 4, so the total relative distance from the lower bound of the resulting solution will be

$$\begin{aligned}
\delta_G <& n\left[\frac{1}{\mathcal{A}_p}+\frac{1}{\sqrt{\mathcal{A}_p}}\right] \\
&+\frac{4(\mathcal{A}(\mathcal{G})-\sum_{i=1}^{n}P_i\mathcal{A}_p)-2(P-\sum_{i=1}^{n}P_i)\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil}{2P\left\lceil 2\sqrt{\mathcal{A}_p}\right\rceil} \\
<& n\left[\frac{1}{\mathcal{A}_p}+\frac{1}{\sqrt{\mathcal{A}_p}}\right]+\frac{\mathcal{A}(\mathcal{G})-\sum_{i=1}^{n}P_i\mathcal{A}_p}{P\sqrt{\mathcal{A}_p}} \\
=& n\left[\frac{1}{\mathcal{A}_p}+\frac{1}{\sqrt{\mathcal{A}_p}}\right]+\left[1-\frac{\sum_{i=1}^{n}P_i\mathcal{A}_p}{\mathcal{A}(\mathcal{G})}\right]\sqrt{\mathcal{A}_p}
\end{aligned}$$

∎

Chapter 3

The Snake Decomposition Algorithm

3.1 Overview

In light of the theorems of the previous chapter, it becomes apparent that under certain assumptions, a 5-point uniform grid can be decomposed such that its total perimeter is relatively close to the lower bound. The `stripeFill(...)` routine that we presented can be extended so as to provide a general algorithm for partitioning *any* 5-point grid. In theorem 14, stripe decomposition is performed on the largest embedded rectangles of the grid, and then a very simplistic approach is taken to complete the cell assignment (namely, scan all the grid cells and assign each unassigned cell to a processor that hasn't been assigned $\mathcal{A}_p$ cells yet). *Snake decomposition* is an algorithm that extends the idea of partitioning a rectangular grid into stripes of optimal height.

3.2 The Snake Decomposition Algorithm

To describe the snake decomposition method, observe that any 5-point uniform grid can be represented by an $M \times N$ rectangle with some of its cells having a certain value to indicate that they are not part of the grid, or "unavailable". This super-grid is the smallest rectangular grid that can accommodate our given grid; in the combinatorics literature [Mel94], this rectangle is sometimes called the convex hull of the grid. Snake decomposition accepts as input a

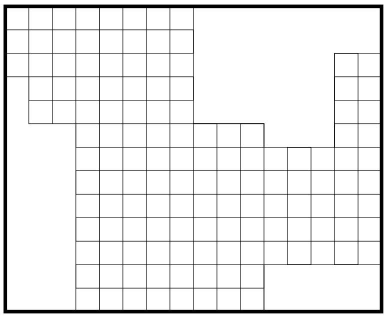

Figure 16: Rectangular Hull Representation of a Grid

partition of the rows of this super-grid (see the thick rectangle in figure 16) into a set of near optimal heights, and then fills the grid with the required number of processors by filling the columns of each stripe consecutively in a way that resembles the movement of a snake. The first stripe's columns are filled left to right, then the second stripe's columns are filled going right to left, and then the process repeats. If the end of a stripe has been reached, but the processor that was used to fill the last part of it still hasn't been assigned $\mathcal{A}_p$ cells, then part of the stripe(s) immediately below the area that the current processor occupies are assigned to this processor in a row-by-row fashion. This is done in order to keep the perimeter small, since otherwise the processor will have to get cells totally unconnected to the ones that are already assigned to it, which is very likely to significantly increase its total perimeter. A pseudo-code for this algorithm follows:

```
snake(int G[M,N], stripe[S], P)
/* Inputs: The grid G is an MxN array;                        */
/*       : stripe[S] is an array of S heights adding up to M */
/* Output: An assignment of G among the P processors          */
begin
proc = 1;
stripeIndex = 1;
start = 1;
done = FALSE;
areas = computeAreas(area(G), P);
while not(done)
  fin = start + stripe[stripeIndex];
  /* left to right */
  fillLeftToRight(G, start, fin, proc, areas, new, done);
  /* done with? */
  if (done) end
  endif;
```

```
  if (not new)
    nextEnd = fin+1+stripe[stripeIndex+1];
    /* finish current */
    finish(G, fin+1, nextend, proc, areas)
  endif;
  start = start + stripe[stripeIndex];
  stripeIndex = stripeIndex + 1;
  fin = fin + stripe[stripeIndex];
  /* right to left */
  fillRightToLeft(G, start, fin, proc, areas, new, done);
  /* done with? */
  if (done) end
  endif;
  if (not new)
    nextEnd = fin+1+stripe[stripeIndex+1];
    /* finish current */
    finish(G, fin+1, M, proc, areas)
  endif
endwhile
end;
```

Routine `fillLeftToRight(...)` accepts as input the grid $\mathcal{G}$ (with its partial assignment), the `start` and `fin` rows of the stripe it will work on, and the current processor index `proc`; it outputs the variable `done` which indicates whether the assignment of processor indices to the grid cells has completed (in which case the algorithm terminates) and the variable `new` which indicates whether the current processor must continue on the next stripe. The routine starts from the leftmost column of the stripe and assigns unassigned cells in each column (scanning it top to bottom), with the current processor index and immediately decreasing the area of the processor by one. As soon as the area of the current processor becomes zero, the current processor index is increased by one. The routine `fillRightToLeft(...)` works identically except that it works its way from the rightmost column of the stripe and moves to the left end. A pseudocode for these routines follows:

```
fillLeftToRight(int G[M,N], start, fin, p, areas[P],
                boolean new, done)
/* Inputs: The grid G is an MxN array;                    */
/*       : start, fin are stripe starting and ending rows  */
/*       : p is the current processor index                */
/*       : areas holds the remaining area for each proc    */
/* Output: new indicates whether a new processor is used   */
/*       : done indicates termination                      */
/*       : An assignment of the stripe among processors    */
begin
/* go left to right */
```

```
for j = 1 to N
  for i = start to fin
    if (p < P and areas[p] = 0)
      /* increase current processor index */
      p = p+1;
      /* new processor starts */
      new = TRUE
    endif;
    /* free cell? */
    if (G[i,j] = 0)
      G[i,j] = p;
      areas[p] = areas[p] - 1;
      new = FALSE
    endif
  endfor
endfor;
if (areas[P] = 0)
  done = TRUE
endif;
end;
```

The routine `fillRightToLeft(...)` is identical to the above routine except for the first statement which reads

```
/* go right to left */
for j = N to 1
```

The routine `finish(...)` accepts as input the grid $\mathcal{G}$, the starting row of the next stripe `fin+1` and completes the assignment of the current processor `proc` by continuing into the immediate rows of the next stripe.

```
finish(int G[M,N], start, end, proc, areas[P])
/* Inputs: The grid G is an MxN array;                   */
/*       : start is the beginning row of the next stripe */
/*       : end is the ending row of the next stripe      */
/*       : proc is the current processor index           */
/*       : areas[P] is the remaining areas for each proc */
/* Output: Completes the assignment of the current proc  */
begin
  s = startCol(proc, start-1, G);      /* get starting col */
  e = endCol(proc, start-1, G);        /* get ending col */
  for j = s to e
    for i = start to fin
      if (areas[proc] > 0 and G[i,j] = 0)
        G[i,j] = proc;                 /* assign free cell */
        areas[proc] = areas[proc] - 1  /* decrease area */
      endif
      else if (areas[proc] = 0)
```

```
        end
      endif
    endfor
  endfor
end;
```

The runtime of the algorithm is linear in the size of the grid since each grid cell may be examined at most twice; if a cell is set in the `fillLeftToRight(...)` or `fillRightToLeft(...)` routines then it is never examined again; otherwise, if it is set from within the body of the `finish(...)` routine, the next `fillRightToLeft(...)` or `fillLeftToRight(...)` call will examine it (and leave it intact). So the runtime of the algorithm, for any input array `stripe[P]`, is $O(|\mathcal{G}|)$ which makes it a very fast routine.

Figure 8 shows an optimal partition (i.e. , one that matches the lower bound) of a 200×200 rectangular grid among 200 processors (obtainable by stripe or snake decomposition), while figure 17 shows a partition of a circle into 64 equi-area sub-domains obtained by snake decomposition. The horizontal lines that are formed in both figures at the boundaries of the partitions correspond to the various stripe heights. The partition of the circle has an associated relative gap of 5.87% from the lower bound, the best found by any method we have tried.

The main difference of the two methods lies in the fact that snake decomposition allows a stripe to "overflow", i.e. the last processor used in a stripe is allowed to "continue" over the stripe immediately below (see the shapes of the processors that occupy the ends of the stripes in figure 17).

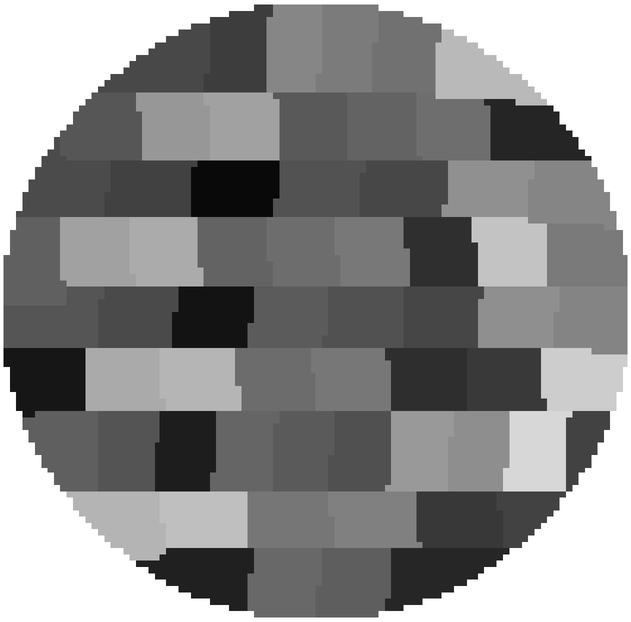

Figure 17: Snake partition of circle among 64 procs

The linear running time of the procedure `snake(...)` indicates that we should expect very good response times of the algorithm in practical implementations. But our goal is twofold: to find very good quality solutions as fast as possible. The key to obtaining high-quality solutions is the availability of a partition of the rows of the grid that will produce stripes with minimal total perimeter. The response time of the algorithm will be about the same for any such partition of rows, but the quality of the produced solutions can vary dramatically. In fact, solutions of high quality might actually be produced faster because of fewer double examinations of grid cells (in high quality solutions only a few processors "overflow" in the next stripe, so few double cell examinations are incurred).

The next chapter examines *Genetic Algorithms* (GAs) as a means for obtaining good input stripes to pass to the snake decomposition algorithm.

Chapter 4

Parallel Genetic Algorithms

4.1 Overview

Snake decomposition, as described in chapter 3, accepts as input a partition of the rows of the grid and subsequently computes a partition of the grid among the required number of processors. However, the number of possible partitions of the rows of the grid increases rapidly as M, the total number of rows increases. In particular, it is easy to show that the number of feasible partitions of M into k components is $n(k, M) := \begin{pmatrix} M-1 \\ k-1 \end{pmatrix}$ where a partition of M into k components $(x_1, \ldots, x_k)$ is feasible if $\sum_{i=1}^{k} x_i = M$, with $x_i \in \mathbb{N}^*$. Therefore, the total number of feasible partitions of M is

$$R(M) := \sum_{k=1}^{M} n(k, M) = 2^{M-1}$$

which grows exponentially fast.

To search the huge space of input partitions to the snake decomposition routine, we use the *Genetic Algorithm* paradigm. Genetic Algorithms (GAs for short) are a class of randomized algorithms that are inspired by the evolutionary processes of nature, and attempt to find optimum or near optimum solutions to a problem by breeding a population of solutions; this population is driven towards better solutions by a process that resembles "natural selection", the mechanism that is responsible for the evolution of species according to C. Darwin [Dar59]. Although the origins of this class of algorithms might be traced back to the early '50s (according to Michalewicz in [Mic94]) it was John Holland [Hol92] who pioneered the field and developed a theoretical framework for the use of GAs. In the subsequent sections, we briefly discuss traditional GAs and the models of generational replacement and the steady-state approach, and then describe in detail a new distributed, fully asynchronous GA (DGA) that is based on the *island* model. The results of this GA are superior to any other method we have tried for graph-partitioning.

4.2 The GA Model

A GA is a randomized algorithm that breeds a –usually fixed– population of *individuals* which are represented by strings of an alphabet –traditionally the binary alphabet $\mathbf{B} = \{\mathbf{0}, \mathbf{1}\}$– which themselves represent a possible solution to a given problem. Crucial to the success of the Genetic Algorithm is the existence of a *fitness function* that can be applied to any individual to produce a reasonable metric of the quality of the solution the individual represents for our problem. This metric is called the *fitness value* of the individual. Assuming the existence of such a function, the algorithm bootstraps its computations by creating a random initial population of individuals, and then proceeds in generations where at each generation some of the most fit individuals are selected for mating and form couples that exchange parts of their strings to form a new individual, a process called *crossover*. Other genetic operators such as mutation or inversion may be applied to the newly created individual and then this newborn offspring is evaluated using the fitness function. Then, according to a survival policy the individual might or might not replace one of its parents in the population. A pure survival policy will always replace one of the offspring's parents with the offspring no matter what the fitness value of the offspring is. More elitist strategies will discard the offspring in favor of the parents if they are more fit than their children. The algorithm stops when some termination criterion has been met, like a provably optimal solution has been found, or a maximum number of generations has been reached, or the population has converged to a point from which no further improvement can be made.

The above general procedure shown in figure 18 leaves many decisions to be made about the specific workings of the GA. In fact, even the string representation of the solutions can be altered or generalized to include other more appropriate data structures for the original problem or simply more letters in the original alphabet, and this is some times the case in discrete optimization problems like the QAP or the traveling salesman (TSP) problem (see [Mic94] for more details on representation issues for GAs).

- *Genetic Operators:* Having decided on the representation of the solutions,

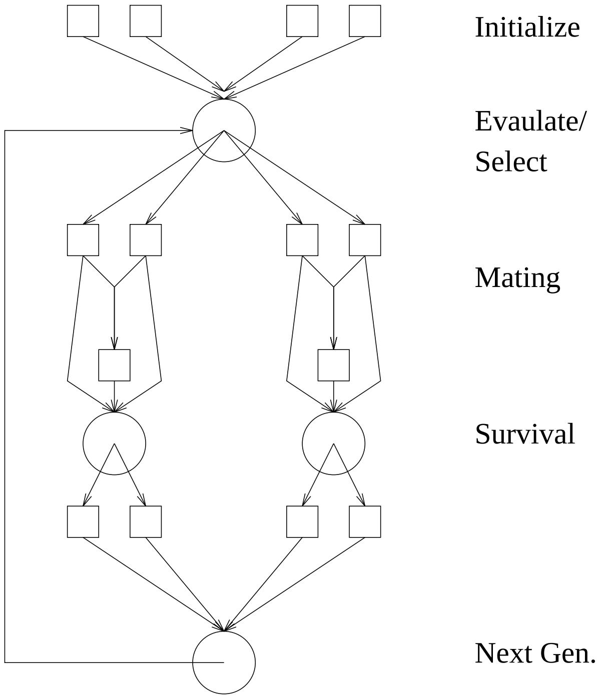

Figure 18: The GA model

the next task is to choose the genetic operators to be used for the creation of the next generation. The most widely used ones are one-point or two-point crossover (binary operators), mutation, and inversion (unary operators) but quite often special purpose operators might be needed to enhance performance or to ensure the feasibility of the solutions represented by the offspring.

- *Survival Policy:* The survival policy is a set of rules that determines when an offspring will replace one of its parents in the next generation, or it will be discarded. In general, the more elitist a policy is (keeping the best individuals in the population) the faster the whole population converges to a homogeneous state, where every individual is similar to every other, and with approximately the same fitness value. The problem is that usually fast convergence locates only a local optimum of moderate quality.

- *Selection:* Another decision that has to be made concerns the process that selects individuals for mating. Several such routines have been proposed. Among the most successful ones are tournament selection, roullete wheel based and remainder stochastic sampling with replacement. These policies try to select individuals for mating based on their fitness values in such a way so as to strike a good balance between selective pressure (the force that drives individuals towards promising regions of the search space) and population diversity (for the exploration of as much of the search space as possible).

- *Steady State vs Generational Replacement:* Members of the current population might not be selected for mating but still continue their existence in the next generation. This policy is known as the steady-state approach, where in each generation only a small percentage of the population is selected for mating and their offspring replace the parent individuals. The traditional GA follows the generational replacement model, where the whole population is replaced by their offspring in the next generation.

- *Parameter Setting:* Finally, when all of the above decisions have been made, the appropriate parameters must be set (cross-over and mutation rate, population size etc). Choosing a set of parameters is usually done via extensive experimentation.

The GA paradigm offers an excellent compromise between the themes of exploration and exploitation. Problems with poor structure (such as the NP-hard problems) have to be tackled with search methods that are capable of exploiting promising regions of the search space as well as exploring in some reasonable way the rest of the space and if they are to be of any practical use, manage to accomplish both tasks without facing combinatorial explosion (else brute force will explore all of the space and come up with the provably optimal solution to any problem –the key is to obtaining high quality solutions within an acceptable time frame).

There is a lot of literature arguing why GAs indeed manage to strike a good balance between the two almost orthogonal goals of exploration (widening the search horizons) and exploitation (narrowing the focus of the search to locate high quality solutions). The basic idea is that for problems that satisfy the *building block hypothesis*, that is, for problems where parts of two different strings representing different solutions, can be combined to produce a better solution, the GA model can be expected to find high quality solutions. The schema theorem [Hol92] formulates the above statement mathematically and proves its validity.

4.3 Handling Constraints

Even though GAs perform remarkably well in many optimization-related fields, heavily constrained problems in general can be hard for a GA to solve unless the structure of the feasible region can be somehow exploited and if possible embedded in the representation. The basic difficulty a GA will have in the face of a constrained problem is that often the standard genetic operators crossover and mutation will produce infeasible offspring (see [Mic94] for an extensive treatment on the subject). There are three main techniques for dealing with this problem.

- The first technique is most appropriate when one can exploit the structure of the problem. It incorporates the constraints in the encoding of the individuals in such a way so that the genetic operators that will be applied to any two valid individuals can be guaranteed to produce offspring that also represent feasible solutions. Whenever such a construction is possible it is worthwhile pursuing, as it is probably the best and most natural solution to the problem; unfortunately often the task of incorporating the constraints in the representation is far from trivial.

- The second technique is based on penalty functions –a topic well studied in optimization– and attempts to assign some fitness value to infeasible individuals depending on how close they are to the feasible region. In heavily constrained problems (such as discrete optimization problems like traveling salesman, scheduling, or network flows) however, even if the GA is initialized with a feasible population, after a few generations, the population will have very few, if any, feasible individuals, the reason being that recombination of solutions will tend to produce infeasible individuals since the feasible region consists of very narrow areas, and so progress (finding a better solution) will be very difficult.

- Finally, the third technique for handling constraints, called repair method, uses a repair algorithm that, given an offspring, repairs it by adjusting as few alleles (positions in the chromosome) as possible so that the resulting individual represents a feasible solution. Repair algorithms offer a compromise between incorporating the constraints in the representation and the operators, and the difficulty of this approach, by allowing representations and operators to produce infeasible offspring and then modifying these offspring to valid individuals. (If the repair techniques can be moved to the fitness function, thus allowing individuals that might otherwise appear infeasible become now feasible, the change may be considered a representation change.)

4.4 Parallel GA Models

Another very important aspect of GAs is the fact that they are by their nature, parallel algorithms. Working with whole populations of individuals rather than with a single individual at a time, parallel computing environments can be used to speed up the computations by assigning a number of individuals to each processor and letting the process of fitness evaluation proceed in parallel. Computationally time-consuming fitness evaluations (such as done for large equi-partition problems) are particularly well-suited for parallel computation.

Often, each processor, after computing the fitness values of its individuals, sends the results back to a co-ordinator master processor that decides (based on the selection policies it has) which individuals mate, and then communicates the answer back to the nodes. After the nodes receive the decision from the host and once they acquire the individuals they need for the mating to proceed, they begin another iteration (computing the offspring and evaluating them). In this scheme, co-ordination is needed, the master processor bases its decisions on global information about the whole population, and the programming style is a host-node synchronized communication-computation (see figure 19). Similar versions of this model have been implemented by various researchers –see [vL91, CMY93, CM96c].

In another more recent approach [MSB91, Lev95], each processor is considered an (almost isolated) island which periodically sends qualified individuals to another island according to some criterion. In this case, the whole GA process is carried out locally without global knowledge of what the populations in the other processors (islands) are doing (with the exception of a periodic *migration* of individuals). The advocates of this theory argue that the new model

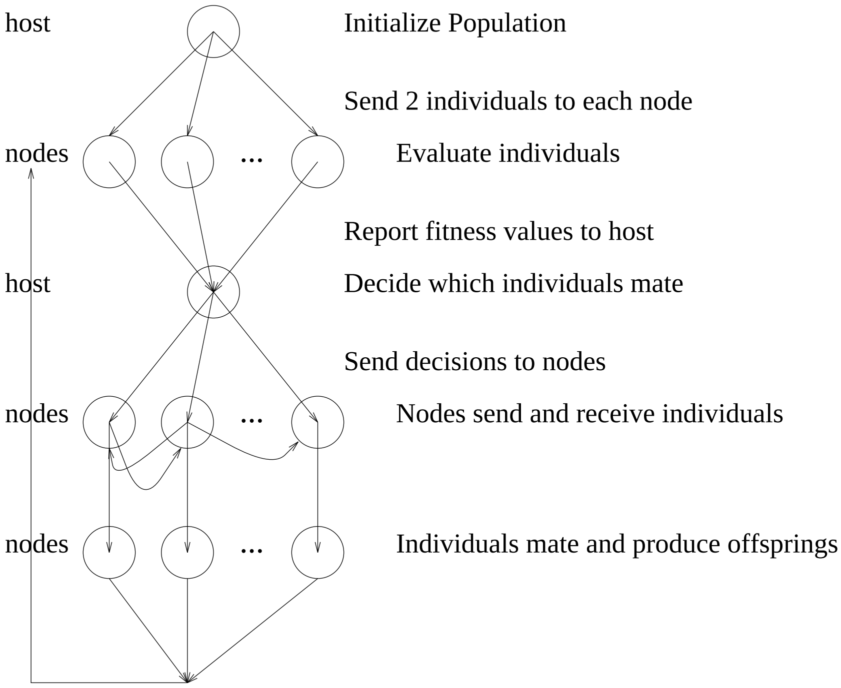

Figure 19: Host-Node Parallel GA

helps avoid the phenomenon of premature convergence, a phenomenon where early in the evolution process the population is driven in a local optimum and becomes nearly homogeneous; after this has happened, the evolution process cannot escape the local optimum because not enough diversity (genetic as well as schematic) is present to allow the genetic operators to find better quality solutions. Because premature convergence is often due to the early appearance of a super-individual, that is an individual whose fitness value is far above the average, which creates many offspring and thus drives the rest of the population towards it (within a few generations the population consists of descendants of this super-individual), having many islands that are isolated helps avoid this problem. Appearance of a super-individual no longer affects all of the population and genetic diversity is not lost. So the themes of exploration and exploitation may both continue. Furthermore, programming this model, allows for a fully asynchronous distributed GA that periodically exchanges information between its processes (islands). A schematic figure of this model is shown in figure 20.

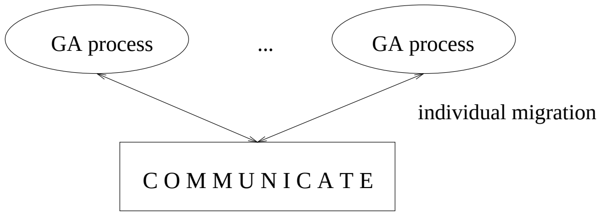

Figure 20: Asynchronous Distributed GA Communication

4.5 DGA: a New Distributed GA

Genetic Algorithms traditionally operate on a fixed population of individuals, using the general schemes described above. DGA is a new, fully asynchronous, distributed Genetic Algorithm following the island model of computation. When started, DGA spawns a certain number of processes (each of which is called an *island*) on the platform on which it is running, and each island is initialized with a certain number of individuals created randomly. It uses the steady state approach as the mechanism for creating the next generation of individuals, but it also includes a new feature, namely age (to be discussed below), in order to control the population and to maintain diversity of the population and prevent premature convergence.

Since our goal is to develop efficient algorithms and data structures for solving the minimum perimeter problem (MP), we have chosen the most natural representation of a partition of the rows of the grid for the GA: the string representation. However, each allele in the chromosome is not a binary number, but rather a natural number greater than zero that represents stripe height. This implies a modified design: each individual in the population may have different chromosome (DNA) length as long as it represents a valid solution, i.e. a partition of the rows of the grid. Furthermore, the age mechanism suggests a policy that allows each DGA island to have a variable population size, Figure 21 shows an illustration of the workings of this new GA.

A main iteration in each island performs the following steps each of which will be explained in more detail below:

- For a certain number of times select two individuals, apply crossover and mutation to create a new offspring, evaluate it and place it in the island (replacing another individual following an elitist strategy if there is no empty space in the island).

- Increase the age of each individual and remove “olds” according to a probability distribution (specifically, the normal distribution).

- Broadcast the island’s population.

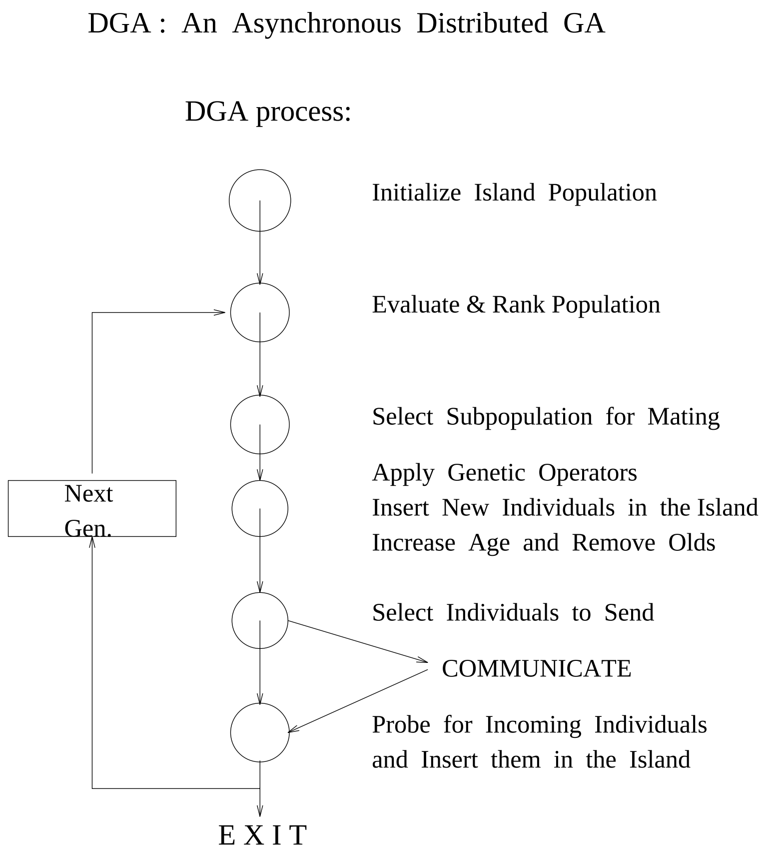

Figure 21: Asynchronous Distributed GA

- If migrating criteria are satisfied, select qualified individuals for migration to other islands and send them.

- Probe for incoming individuals and receive all that have arrived.

The algorithm is fully asynchronous in that all send and receives are done asynchronously, i.e. when a process sends a message to another process, it does not wait until the receiving process receives the message. Instead, it resumes computation as soon as the outgoing message is safely on its way on the network.

The criteria for sending individuals to other islands are based on workload values: if an island's population has dropped too much then a decision is made to send the best individuals from another island to the depopulated island.

All processes during the probing step simply probe their buffer pool to check for pending messages, and if there are any, they are received, else computation is resumed from the point it was left. Thus, an individual sent to a process during an early generation might actually get there several generations later. However, the approach offers the advantage of not wasting network and CPU resources waiting for ACK/NACK signals, which in an environment like a network of workstations can make a difference. In the probing step, any incoming individuals are ranked according to their fitness value and inserted in the population.

Indirectly, the criterion for sending individuals to other islands helps increase the average fitness of the depopulated islands, because islands become deserted when individuals are deleted because of old age (migration should not affect an island's population too much because only islands that are very crowded may send individuals to other islands). But the age at which an individual is deleted is proportional to its fitness value and, thus, one would expect that islands with well fit individuals never become deserted while islands where the average fitness is low tend to shrink in population size, and thus become fertile ground for highly fit individuals to migrate. To the author's knowledge, such a use of an "age" mechanism in an island model of a GA has not been studied before.

The age mechanism also helps maintain diversity of the population within each island by eventually removing any individual no matter how well fit. This prevents an elitist strategy from finding an individual representing a good but not even near optimal solution and then driving the whole population towards this solution, resulting in premature convergence. Of course, this also means that the population will converge much more slowly to a homogeneous state (if at all). But our goal is not to observe population convergence; we are mainly interested in finding high quality solutions to the MP problem, and thus we keep an incumbent individual for each island (which records the best individual ever found on this island) and our termination criterion is a fixed number of generations. With careful experimentation and parameter setting, we have been able to find near optimal –and often provably optimal– solutions to the MP problem as will be evident in the chapter describing the computational results.

As already mentioned, we have introduced variable chromosome length individuals because our solution space consists of integer arrays (collections of stripe heights) of various lengths with the property that the sum of each array's components is fixed (and equal to the number of rows of the grid to be partitioned). One can imagine many other applications where varying chromosome lengths are best for representing the search space.

Chapter 5

Computational Results

5.1 Overview

In this chapter we present the computational results of our partitioning algorithm, namely DGA, having as fitness function the snake decomposition routine, and make comparisons with other well established codes for graph partitioning. We have used two classes of domains: rectangular grids, and non-rectangular grids (that is, uniform but irregular boundary 5-point grids). COW is a cluster of 40 high-performance Sun SPARC Server 20 workstations running Solaris at the Computer Sciences Dept. at the University of Wisconsin - Madison. Our experiments were performed on the nodes of this network (in [CM96c] we report results from runs of another GA on a CM-5 with 32 nodes).

We compare the performance of DGA against that of recursive spectral bisection (RSB) and quadrisection (RSQ) as implemented in the Chaco package [HL95a], a well-known package for graph partitioning developed at Sandia National Laboratories; on the class of rectangular grids we also make comparisons with the geometric mesh partitioner [MTTV93, GMT95], another well known mesh partitioner and a state-of-the-art GRASP heuristic for the QAP [LPR94] that was written at AT&T Bell Labs. Finally, we test DGA against R-SNAKE to check whether a Genetic Algorithm outperforms random selection in finding good inputs to the snake procedure, and we also make comparisons with PERIX-GA [CM96b, CM96c], a GA following the host-node programming paradigm using generational replacement with a fitness evaluation function that is based on a different tiling approach. PERIX-GA is a synchronous GA running on a 9-node partition of the COW. PERIX-GA assigns 2 individuals per processing node and a host processor co-ordinates the evolutionary process.

5.2 Settings

R-SNAKE was written in ANSI C, compiled with the -O2 optimization option and run on a Sun SPARC Server 20 workstation running the Solaris operating system. The same holds true for the Chaco package. All experiments with Chaco had the Kernighan-Lin post-processing phase option turned on. The implementation of the geometric mesh partitioner that was available to us was written in MATLAB, which partly explains the very long times of the method for many of the test problems. The GRASP code for the QAP was written in FORTRAN 77 and compiled with the -O optimization option; it run also on a Sun SPARC Server 20 workstation. It was allowed to run for 100 iterations. DGA was written in ANSI C, and utilizes the PVM 3.3.10 message passing interface for interprocess communication [GBD+94]. DGA runs on the COW. PERIX-GA was also written in ANSI C and uses the PVM 3.3.7 calls for message passing on the COW (and the CMMD libraries for message passing on the CM-5). PERIX-GA was allowed to run for 20 generations.

R-SNAKE was allowed to try 100 random valid stripe arrays and it reports the best partition found. DGA requires more parameters. Each DGA island (process) maintains a maximum of 16 individuals per generation. There are 4 and 8 islands spawned (using the hostless programming paradigm). Whenever an individual arrives at an island, it replaces the worst individual in the population if there is no empty space in the island by the time it arrives. One-point crossover was used, with the rate set at 0.85. The mutation rate (defined here as the probability of mutating an allele of the chromosome) was set at 0.15. Since the product of the crossover and mutation operators may result in something that is not a legal individual, a *repair strategy* was used. Each new individual is given as input to a repair routine that modifies as few alleles as possible (sometimes even changing the length of the individual) so that the resulting individual is legal, i.e., represents an exact partition of the rows of the grid. Finally, as the algorithm follows the steady-state approach, approximately 70% of the populations new individuals are born in each generation (not necessarily replacing an equal number of old individuals, because whenever empty space exists, a new individual is inserted without replacing any existing individual). Each individual's lifespan is determined as a random variable that follows the Gaussian distribution $\mathbf{N(5 * fitness(individual), 0.09)}$. DGA was allowed to run for 20 generations. Experimentation showed that the above parameters resulted in good overall GA behavior. More details on the experiments conducted with DGA can be found in [CM96a].

5.3 Rectangular Grids

In table 1 we give the size of the rectangular problems in our test-suite using a QAP, linear MIP, or a GA formulation. In the QAP literature, the dimension of a problem is the number of rows (or columns) of the distance (or cost) matrix of the problem. Thus, for the QAP formulation of the MP we are using, the QAP dimension of a problem is equal to the number of grid cells in the graph. In the GA formulation, the size of the problem is measured as the number of the required partitions.

PROBLEM			QAP	MIP		GA
M	N	P	DIMENSION	VARS	CONSTR	VARS
7	7	7	49	427	3584	7
32	31	8	992	9857	108576	8
32	31	256	992	255873	125404128	256
32	30	64	960	63298	7492480	64
100	100	8	10000	99800	1118808	8
128	128	128	16384	2129664	5.285E+08	128
256	256	256	65536	1.690E+07	8.523E+09	256
512	512	512	262144	1.347E+08	1.369E+11	512

Table 1: Problem Sizes under Various Formulations

In table 2 we compare recursive spectral bisection and geometric mesh partitioner against GRASP for the QAP. The times on all tables are in seconds. The columns labeled "Gap" show the relative distance of the solution from the

PROBLEM		RSB		GEOMETRIC		GRASP	
$M \times N$	P	Time	Gap%	Time	Gap%	Time	Gap%
7 x 7	7	-	-	-	-	182.9	0.00
32 x 31	8	1.8	6.52	43.6	5.43	-	-
32 x 31	256	4.3	6.73	152.3	-2.73*	-	-
32 x 30	64	3.0	6.25	90.4	6.25	-	-
100 x 100	8	9.0	9.33	111.0	7.39	-	-
128 x 128	128	85.5	14.13	539.9	7.13	-	-
256 x 256	256	227.8	13.25	3304.2	4.15	-	-
512 x 512	512	-	-	-	-	-	-

Table 2: Spectral Bisection, Geometric and GRASP

lower bound. An asterisk in table 2 indicates the fact that the partition found was not balanced (i.e. there were components that had at least two more nodes than other components). For the first problem, since the number of partitions required is not a power of 2, neither RSB nor the Geometric Mesh Partitioner could solve it. Also, note that for the last problem, both the geometric and the spectral method ran out of memory when trying to construct the adjacency matrix of the graph. It is apparent that even for small grid graphs, the QAP approach to solving the MP has difficulties because in the QAP literature, problems with dimension higher than 30 are considered challenging problems. (An experiment was performed to solve the MP(13×13, 13) problem on a Sun SPARC-Station 10 with 64MB of RAM, and it took GRASP more than $10,000$ seconds to come up with a solution that was more than 25% away from the lower bound which is actually attainable for this problem.)

As we can see from table 3, PERIX-GA outperforms the programs in table 2 in solution quality, but takes longer to finish on the smaller grids. On the other hand, R-SNAKE and DGA, significantly outperform PERIX-GA in response time, and find the same quality or better partitions (and consequently outperform the spectral and the geometric method as well) as is evident from table 3.

PROBLEM		PERIX-GA		R-SNAKE		DGA 4procs		DGA 8procs	
$M \times N$	P	Time	Gap%	Time	Gap%	Time	Gap%	Time	Gap%
7 x 7	7	15.4	0.00	0.8	0.00	7.4	0.00	8.3	0.00
32 x 31	8	84.0	2.71	1.3	1.08	6.9	1.08	9.2	1.08
32 x 31	256	80.4	0.00	0.8	0.00	7.1	0.00	10.1	0.00
32 x 30	64	50.9	0.00	0.5	0.00	7.8	0.00	9.4	0.00
100 x 100	8	81.9	2.64	2.4	2.28	17.7	2.28	20.4	2.28
128 x 128	128	67.6	1.65	3.5	1.90	15.5	1.63	16.5	1.63
256 x 256	256	105.1	0.00	16.9	0.00	36.9	0.00	38.5	0.00
512 x 512	512	279.0	1.63	58.7	0.68	123.8	0.56	103.7	0.63

Table 3: PERIX-GA on 9-procs vs. R-SNAKE and DGA on 4 or 8 procs

As can be easily seen from tables 2 and 3, R-SNAKE significantly outperforms recursive spectral bisection and quadrisection as well as the geometric mesh partitioner in solution quality, and in most cases in response time as well on all the rectangular domains we tested.

5.4 The Knapsack Approach

For the rectangular domains, the MSP algorithm [Mar96] mentioned in § 2.4 in chapter 2 was applied to the class of problems MP($N \times N$, N), and MP($10N \times 10N$, N); the MSP algorithm determines all "valid" stripe heights (ones that result in stripes that can be covered exactly with an integer number of processors) and applies a knapsack routine to determine the best stripe form decomposition of the given problem. MSP was written in FORTRAN 77 and runs on a Sun SPARC Station 20 workstation running Solaris.

In figure 22 we show the relative distance from the lower bound of the best stripe form partition of the problem MP($N \times N$, N) for $N = 5 \ldots 1000$. The average best relative distance is 0.7%. It is interesting that 32.6% of the problems were solved to optimality. It is also interesting to see that the relative distance of these solutions from the lower bound decreases as N increases in the fashion predicted in theorem 8.

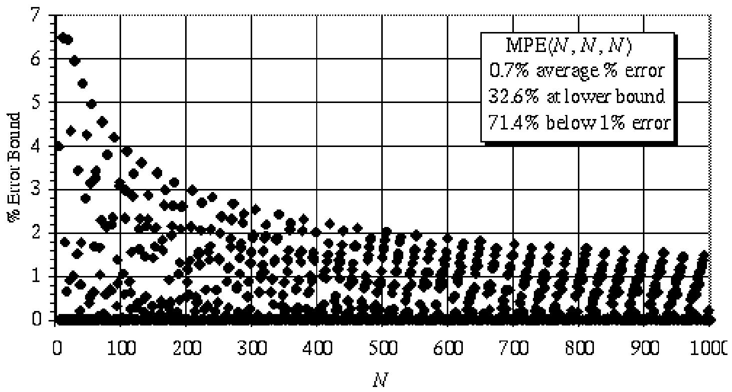

Figure 22: Distance from Lower Bound of Stripe Solutions (N,N,N)

Figure 23 on the other hand, shows the relative distance from the lower bound of the best stripe form partition of the problem MP($10N \times 10N$, N); here the area assigned to each processor is $100N$ and since the number of processors is 10 times smaller than each dimension of the grid, theorem 8 does not apply. However, one can clearly see that the resulting partitions are of excellent quality, so it is tempting to make the conjecture that even if P, the number of available processors does not dominate the dimensions of the grid, under very mild assumptions, the relative distance of the best solutions from the lower bound tends to zero as the problem parameters tend to infinity. For example, the solution obtained for the $10,000 \times 10,000$ grid partitioned among 1000 processors was within 0.042% of the lower bound.

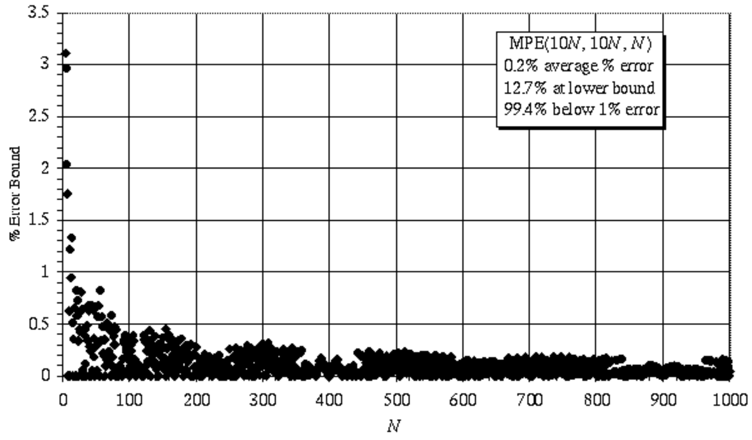

Figure 23: Distance from Lower Bound of Stripe Solutions (10N,10N,N)

5.5 Non-rectangular Grids

On the more difficult irregular boundary problems (for which the knapsack approach is not directly applicable), R-SNAKE and DGA have times that are comparable to the ones given by RSB and RSQ, but the solution quality difference here is more dramatic, rising up to 22 percentage units for the diamond domain partitioned among 16 processors (see tables 4 and 5). The partition of this domain is shown in figure 24. The sole exception to this is the (small) elliptical domain partitioned among 64 processors, where RSB found a marginally better solution than R-SNAKE or DGA.

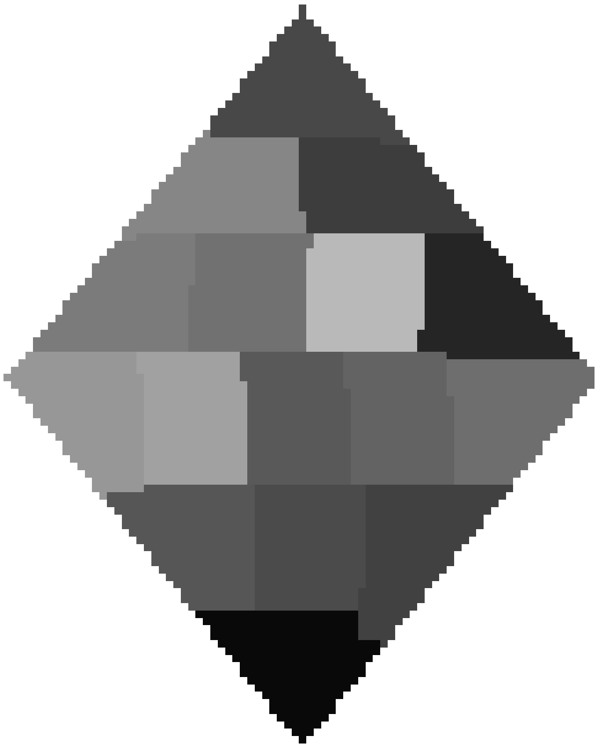

Figure 24: Diamond Domain Partitioned Among 16 Procs

Comparing R-SNAKE with DGA, we observe that DGA finds better (but by no more than 1%) solutions but requires more time. Also, since the times for DGA with 8 islands running on 8 COW nodes (see table 5) are almost the same as those for DGA running with 4 islands on 4 nodes, the communication penalties incurred by our method are minimal compared to other random network factors.

PROBLEM		RSB		RSQ		R-SNAKE	
Shape	P	Time	Gap%	Time	Gap%	Time	Gap%
circle	16	23.3	24.44	9.1	21.80	9.7	8.33
circle	64	34.7	16.87	14.5	28.34	11.7	6.35
ellipse	16	2.3	10.83	1.4	13.33	5.4	8.33
ellipse	64	3.5	5.16	2.2	15.10	4.9	5.56
torus	16	27.3	28.97	12.5	32.67	16.8	11.50
torus	64	36.5	22.86	18.5	34.3	9.9	11.08
diamond	16	14.0	38.67	6.5	35.74	14.6	17.70
diamond	64	18.7	29.78	9.0	28.80	13.2	14.60

Table 4: Spectral Methods vs. R-SNAKE on non-rectangular grids

The size of the irregular boundary grids in our test suite varies; the circle has 7800 cells, the torus 7696 cells. The diamond domain is smaller, with 4019 cells, and the elliptical domain is the smallest in our test suite with only 823 cells. A partition of the torus among 64 processors as found by DGA running on 8 processors is shown in figure 25. The torus is a disk of radius 50 with a hole of radius 7 in its center.

PROBLEM		DGA (4 islands)		DGA (8 islands)	
Shape	P	Time	Gap	Time	Gap%
circle	16	19.8	8.33	20.2	8.47
circle	64	19.4	6.21	17.5	5.87
ellipsis	16	8.3	8.33	6.4	8.33
ellipsis	64	9.4	5.36	7.5	5.56
torus	16	18.8	11.50	16.7	11.50
torus	64	17.2	11.08	13.8	11.00
diamond	16	10.7	16.40	10.4	16.40
diamond	64	16.2	13.37	14.7	13.37

Table 5: DGA on 4 or 8 COW nodes on non-rectangular grids

In figure 26 we plot the response times of RSB, RSQ, and DGA with four (DGA4) and eight (DGA8) islands for the various irregular boundary grids

Figure 25: Snake partition of a torus among 64 procs

in our test-suite. The labels on the x-axis indicate the following problems: Figure 27 then shows for the same problems and methods the quality of the

Label	Problem	P
C16	circle	16
C64	circle	64
E16	ellipsis	16
E64	ellipsis	64
T16	torus	16
T64	torus	64
D16	diamond	16
D64	diamond	64

Table 6: Plot Graph Labels

solutions thus produced.

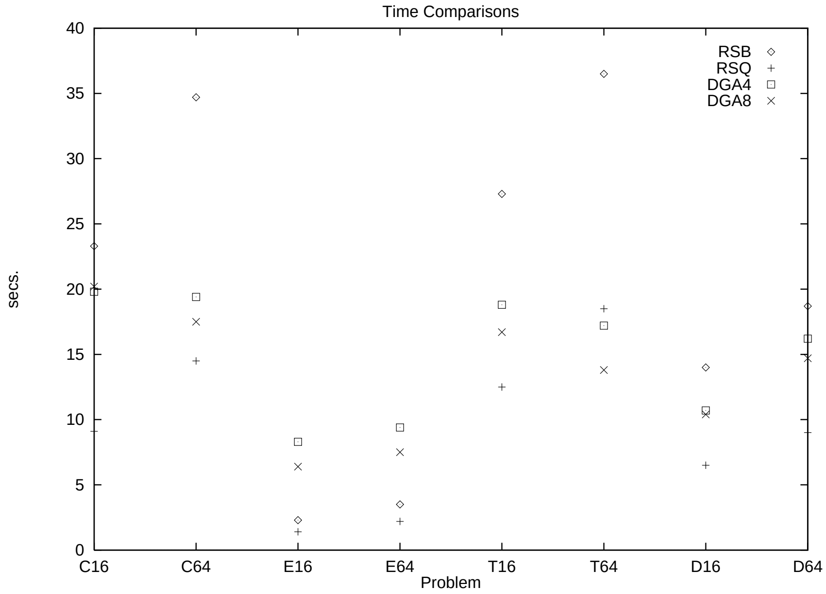

Figure 26: Time Plots for Various Methods on Non-Rectangular Grids

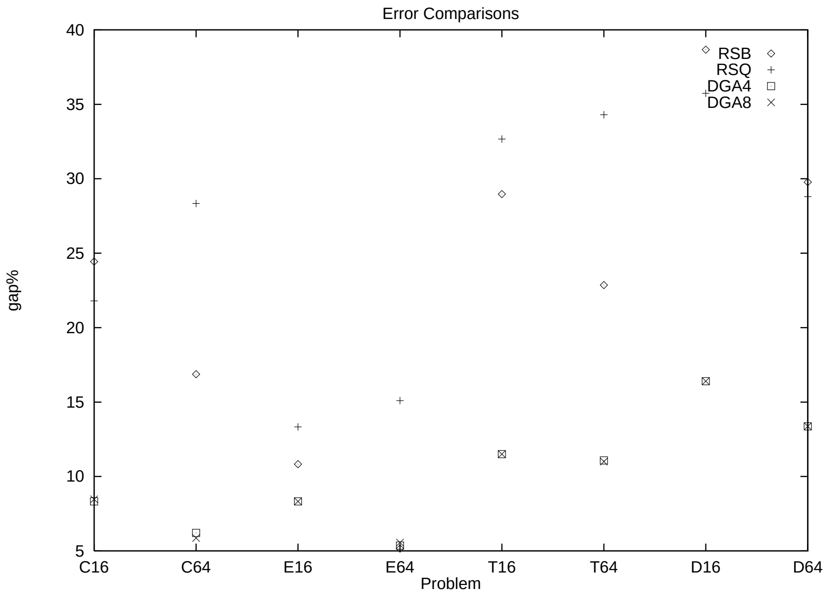

Figure 27: Error Plots for Various Methods on Non-Rectangular Grids

Finally, we run DGA with one island only but leaving all the other parameters intact, disabling all communication (we don't even start PVM) so as to further check the effect of communication on response times of the algorithm. The results (shown under the columns labeled "DGA pop16" of table 7) suggest that indeed the communication overhead of the algorithm is almost negligible. The quality of the resulting partitions is comparable to that of R-SNAKE or the DGA with four (eight) islands, but not as good, and quite logically so, as more processes imply that more individuals are created and tested.

PROBLEM		DGA pop16		DGA pop64		DGA pop128	
Shape	P	Time	Gap%	Time	Gap%	Time	Gap%
circle	16	12.2	9.02	31.8	8.47	60.1	10.27
circle	64	11.3	8.74	31.9	7.03	56.8	6.83
ellipsis	16	1.2	8.33	4.2	9.58	6.6	9.58
ellipsis	64	2.5	5.36	4.7	6.16	7.8	6.75
torus	16	11.4	11.79	30.2	12.78	67.2	12.78
torus	64	11.5	11.08	30.9	11.60	67.2	12.71
diamond	16	5.4	17.18	19.9	16.70	41.5	18.75
diamond	64	7.4	14.06	20.7	13.55	39.9	14.25

Table 7: DGA with 1 island

To isolate the effect of the island model on the evolution process, we also run DGA with 1 island but this time allowing a maximum of 64 or 128 individuals on the island (as opposed to the previous experiments where each island could maintain only up to 16 individuals). As expected, the solution times increase, but what is interesting is the fact that DGA running with a single population that is roughly 4 or 8 times bigger than the population on each of the islands of the previous experiments fails to find the same quality solutions. Occasionally, it beats DGA with one island and a maximum population size of 16, but it never finds the same quality solutions as DGA with 4 or 8 islands. This effect may be partly due to a premature convergence phenomenon: an initial good solution tends to have many offspring during its lifetime thus drives the rest of the population into a homogeneous state that is only locally optimal. The island model helps avoid this phenomenon by keeping separate populations which preserve the overall diversity, which in turn later enables the algorithm to find better solutions.

Chapter 6

Conclusions and Future Directions

6.1 Conclusions

We have presented stripe and snake decomposition, two powerful techniques for equi-partitioning uniform 5-point grids with regular or irregular boundaries among any number of processors, a problem that is NP-hard. We have shown that the application of these techniques to a large class of domains produces solutions that are asymptotically optimal in the sense that as the problem parameters (dimensions of the largest rectangles that fit in the grid and number of processors) grow larger, the relative distance of the perimeter of the corresponding solutions from a theoretical lower bound tends to zero.

To generate even better solutions, we developed a new distributed Genetic Algorithm (DGA) equipped with a snake-filling routine as the fitness function. DGA follows the island model of computation using the steady-state approach for generating the next population in each island. DGA is fully asynchronous so as to avoid communication penalties from a possibly slow network, and to allow for maximum interleaving of computation and communication. Migration criteria are based on the workload of each processor, which is correlated to the average fitness of the populations of each island because of a new mechanism, namely *age*. This mechanism removes an individual from the population when this individual becomes "old". The age at which each individual is removed from the population depends upon its fitness.

The computational results from DGA show that it compares well in response time with other popular graph partitioning methods and that –with one minor exception– produces superior quality partitions for all classes of domains that we tested it.

6.2 Future Directions

While the algorithms we have presented are for uniform 5-point grids, it is possible to extend them to partition non-uniform, non-rectangular domains by recursive application of the snake routine. Non-uniform grids can be thought of as a hierarchy of grids that consist of grid cells some of which are unit cells, and others of which are themselves grids. The snake process can start at the top level of the hierarchy and whenever it finds a "cell" that is itself a grid, it recursively applies itself to the next level of the hierarchy (in the grid contained within this grid-cell) starting with the current processor index. Lower bounds could be obtained by adding lower bounds for each level of the hierarchy, but such lower bounds will not have the simplicity of those considered above, and will not be as close to the optimal value.

Experimenting and evaluating DGA's mechanisms of age and asynchronous migration is another important topic for future research. In particular, it would be very interesting to assess the strength of the age mechanism in the avoidance of premature convergence, and to experiment with new –or variants of the existing– migration criteria that DGA employs.

Our current approach will also be augmented by the inclusion of knapsack methods that generalize the MSP approach described above. While the knapsack approach cannot guarantee the generation of feasible solutions for non-rectangular domains, it can provide good inputs to a repair procedure that will generate feasible solutions by the snake approach. Hence, it offers a promising mechanism for constructing the initial generation in a GA.

Adding a swap phase that considers pairs of cells and swaps them if and only if the swap is not detrimental to the total perimeter of the partition improved the quality of the solutions (at the cost of an increase in response times) in two cases, namely in the torus domain partitioned among 16 processors, and in the diamond domain partitioned among 64 processors. By examining a number of swaps that is equal to 20% of the total area of the grid, DGA produced a partition of the torus that was within 10.65% of the lower bound in 38.2 seconds, and a partition of the diamond that was within 13.18% of the lower bound in 31.3 seconds. For the rest of the problems, pairwise swapping as described did not improve any further the quality of the best solution found. So, instead of relying on pairwise swaps for the final phase of the fitness evaluation, we want to investigate more sophisticated swap cycles based on a Kernighan-Lin procedure [KL70] or linear network approximations to the QAP. More generally, the output of the snake process could be viewed as a starting point for a local search procedure based on interchanges or on more general techniques such as simulated annealing or tabu search.

Extensions of the results to triangulations, three-dimensional domains (where minimum surface area of the partition is one possible objective function), other data partitioning problems arising from parallel database design, and other types of fixed-charge networks also provide promising areas for further research. The same basic idea of fitting together, as well as possible, building blocks that represent optimal solutions to subproblems, applies to these domains as well. Recent results in telecommunications network design and pharmaceutical design indicate that this paradigm of utilizing subproblem solutions within the context of genetic algorithms is very effective in those problem domains as well. With appropriately designed repair procedures for subproblem coordination, we can anticipate that the evolutionary progress associated with genetic algorithms will be effective in producing good solutions for broad classes of large-scale combinatorial network problems.

Bibliography

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows.* Prentice Hall, 1993.

[CM96a] I. T. Christou and R. R. Meyer. Fast distributed genetic algorithms for paritioning uniform grids. In H. Rolim and T. Yang, editors, *Lecture Notes in Computer Science.* Springer-Verlag, 1996. to appear in Irregular 96.

[CM96b] I. T. Christou and R. R. Meyer. Optimal and asymptotically optimal equi-partition of rectangular domains via stripe decomposition. In H. Fischer, B. Riedmuller, and S. Schaffler, editors, *Applied Mathematics and Parallel Computing - Festschrift for Klaus Ritter*, pages 77–96. Physica-Verlag, 1996.

[CM96c] I. T. Christou and R. R. Meyer. Optimal equi-partition of rectangular domains for parallel computation. *Journal of Global Optimization*, 8:15–34, January 1996.

[CMY93] R.J. Chen, R.R. Meyer, and J. Yackel. A genetic algorithm for diversity minimization and its parallel implementation. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 163–170. Morgan Kaufman, 1993.

[CQ95] P. Crandall and M. Quinn. Non-uniform 2-d grid partitioning for heterogeneous parallel architectures. In *Proceedings of the 9th International Symposium on Parallel Processing*, pages 428–435, 1995.

[Dar59] Charles Darwin. *On the Origin of Species by Means of Natural Selection.* J. Murray, London, 1859.

[Fie73] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Math. Journal*, 23:298–305, 1973.

[GBD$^+$94] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM 3 User's Guide and Reference Manual.* Oak Ridge National Laboratory, 1994.

[GMT95] J. R. Gilbert, G. L. Miller, and S. H. Teng. Geometric mesh partitioning: Implementation and experiments. In *Proceedings of the 9th International Symposium on Parallel Processing*, pages 418–427, 1995.

[HL93] B. Hendrickson and R. Leland. Multidimensional spectral load balancing. In *Proc. 6th SIAM Conference on Parallel Processing and Scientific Computing.* SIAM, 1993.

[HL95a] B. Hendrickson and R. Leland. *The Chaco User's Guide Version 2.0.* Sandia National Laboratories, July 1995.

[HL95b] B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM J. on Sci. Comput.*, 16:452–469, 1995.

[Hol92] John Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, 1992.

[KL70] B. W. Kernighan and S. Lin. An effective heuristic procedure for partitioning graphs. *Bell Systems Tech. Journal*, pages 291–308, February 1970.

[Lev95] D. Levine. *User's Guide to the PGAPack Parallel Genetic Algorithm Library Version 0.2.* Argonne National Laboratory, June 1995.

[LFE94] M. Laguna, T. A. Feo, and H. C. Elrod. A greedy randomized adaptive search procedure for the two - partition problem. *Operations Research*, July - August 1994.

[Lin91] K. Y. Lin. Exact solution of the convex polygon perimeter and area generating function. *J. Phys. A. Math Gen.*, 24:2411–2417, 1991.

[LPR94] Y. Li, P. M. Pardalos, and M. G. C. Resende. A grasp for the qap. In P. M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems.* DIMACS Series Vol. 16, American Mathematical Society, 1994.

[Mar96] W. Martin. Fast equi-partitioning of rectangular domains using stripe decomposition. Technical Report MP-TR-96-2, University of Wisconsin - Madison, February 1996.

[Mel94] M. Bousquet Melou. Codage des polyominos convexes et equation pour l'enumeration suivant l'aire. *Discrete Applied Mathematics*, 48:21–43, 1994.

[Mic94] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1994.

[MSB91] H. Muhlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. In R. Belew and L. Booker, editors, *Proceedings of the Fourth Intl. Conference on Genetic Algorithms*, pages 45–52. Morgan Kaufmann Publishers, Los Altos, CA, 1991.

[MT90] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

[MTTV93] G. L. Miller, S. H. Teng, W. Thurston, and S. A. Vavasis. Automatic mesh partitioning. In A. George, J. R. Gilbert, and J. W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*. Springer-Verlag, 1993.

[NW85] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1985.

[PRW93] P. M. Pardalos, F. Rendl, and H. Wolkowicz. The quadratic assignment problem: A survey and recent developments. In P. M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems*. American Mathematical Society, 1993.

[PSL90] A. Pothen, H. D. Simon, and K. P. Liu. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal on Matrix Analysis and Applications*, 11:430–452, 1990.

[Sch89] R. J. Schalkoff. *Digital Image Processing and Computer Vision*. John Wiley & Sons, 1989.

[Str89] John Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. Wadsworth & Brooks, 1989.

[Thi91] Thinking Machines Corporation. *The Connection Machine CM-5 Technical Summary*, October 1991.

[vL91] G. von Laszewski. Intelligent structural operators for the k-way graph partitioning problem. In R. Belew and L. Booker, editors, *Proceedings of the Fourth Intl. Conference on Genetic Algorithms*, pages 45–52. Morgan Kaufmann Publishers, Los Altos, CA, 1991.

[Yac93] J. Yackel. *Minimum Perimeter Tiling in Parallel Computation*. PhD thesis, University of Wisconsin - Madison, August 1993.

[YM92a] J. Yackel and R. R. Meyer. Minimum perimeter decomposition. Technical Report 1078, University of Wisconsin - Madison, February 1992.

[YM92b] J. Yackel and R. R. Meyer. Optimal tilings for parallel database design. In P. M. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 293–309. North - Holland, 1992.

[YMC95] J. Yackel, R. R. Meyer, and I. T. Christou. Minimum-perimeter domain assignment, February 1995. To appear in Math. Programming.