Mesh partitioning & load-balancing for distributed memory parallel systems

C. Walshaw, M. Cross & M. Everett
Parallel Processing Group,
Centre for Numerical Modelling,
University of Greenwich
C.Walshaw@gre.ac.uk

the
UNIVERSITY
of
GREENWICH

Parallel Mesh Partitioning

Wish to solve:

- static partitioning problem
  (the classical problem)
- static load-balancing problem
  (e.g. from parallel mesh generator)
- dynamic mesh partitioning problem
  (e.g. from parallel adaptive refinement)

Strategy

distribute the mesh as quickly as possible (if necessary) and optimise the partition in parallel

Key is **parallel optimisation**

Suggested characteristics:
- single program, multiple data (SPMD)
- multiway (rather than recursive bisection)
- integral load-balancing

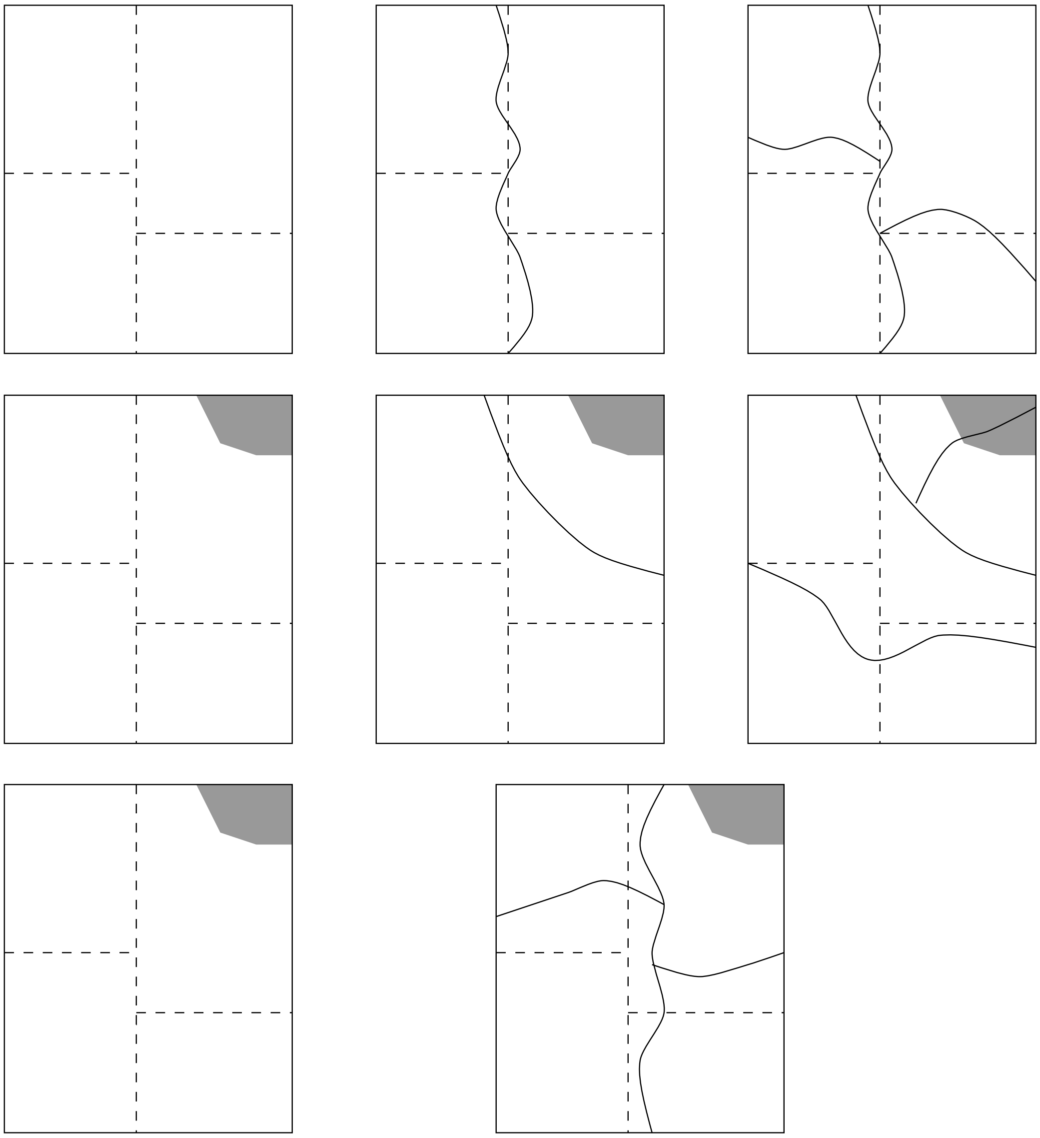
Dynamic Mesh Partitioning

- Synchronous/Quasidynamic
  e.g. Adaptive (de)refinement of mesh
  can interpolate partition to refined mesh
- Asynchronous/Dynamic
  e.g. Multiphysics
    e.g. Casting Simulation
    – solve for flow in liquid/stress in solid

Requirements

- load balance
- reuse existing partition
- parallel algorithm

Dynamic: Multiway vs. Recursive Bisection



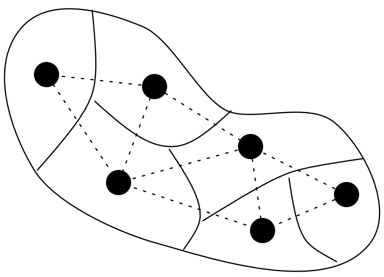
Localisation

Cannot efficiently implement KL/FM algorithm in parallel *(Savage & Wloka, 1991)*

Calculate in border regions only

Only allow transfers:

- of border nodes
- to adjacent processors



Typical subdomain graph

Load Balancing

Calculate flow across edges of subdomain graph.

- Song's algorithm – asynchronous
  (Song, 1994)
- Diffusive – asynchronous
  (Cybenko, 1989)
- Hu's algorithm – synchronous
  (Hu & Blake, 1995)
  - 'implicit' diffusive algorithm
  - rapid convergence
  - minimises norm of transferred weight

Iterative algorithm structure

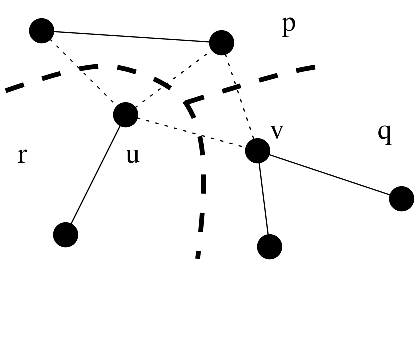
Every iteration

- calculate desired flow across edges
  of subdomain graph
- select border nodes to
  1). Satisfy flow
  2). Maximise gain

Gain & preference functions

Gain $g(v, q)$ expresses the improvement in cost if vertex v moves to processor q.

Preference $f(v)$ is the processor p which maximises $g(v, q)$.



Generic partitioning

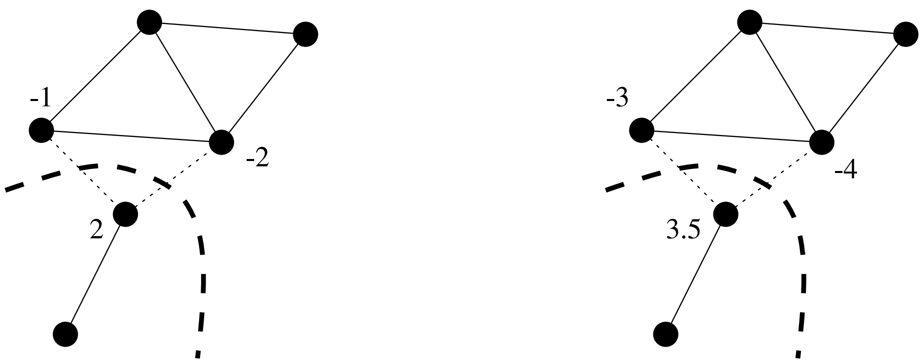
Different gain functions can minimise different cost functions, e.g.

- Cut edges
- Subdomain 'surface energy' *(Walshaw et al., 1995a)*
- Subdomain aspect ratio *(Vanderstraeten et al., 1995; Diekmann et al., 1995)*
- Frontal width *(Vanderstraeten et al., 1995)*
- Hop weight (hypercube) *(Hendrickson & Leland, 1995b)*
- Mapping (processor array) *(Walshaw et al., 1995b)*

Relative gain

Defined as: $g(v,q) - \sum_{u \in \Gamma_q(v)} g(u,p)/\|\Gamma_q(v)\|$

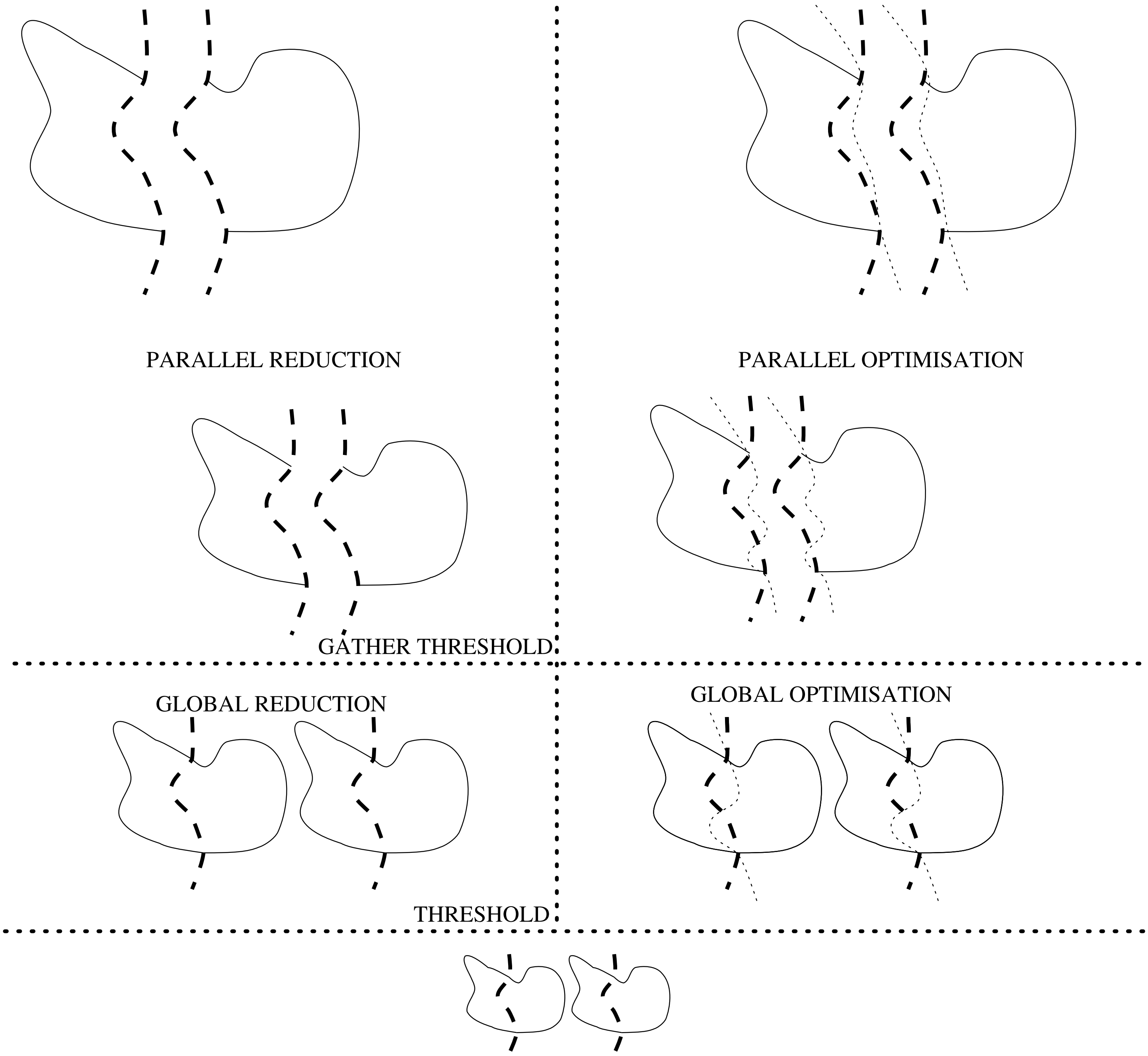
(gain(v) − average opposing gain)



prioritises which vertices to move in parallel

- simple, efficient & parallel
- but not predictable

Parallel Multilevel Optimisation



Graph Reduction

Algorithms:

- random edge matching
  (Hendrickson & Leland, 1995a)
- heavy edge matching
  (Karypis & Kumar, 1995)
- local edge matching
  match with local
  in preference to external

Parallel performance

Reasons for low speedup:

- only integer operations
- only operate on subdomain borders *(in 2D,* $4n \Rightarrow 2\delta n$*)*
- partitioning is neither load-balanced nor optimised
- comms dominate in coarsest graphs

Reasons not to worry:

- partitioning is means to an end
- serial partitioning may be impossible

Static Results on T3E (HLRS - Stuttgart)

- MPI implementation
- serial/parallel partitions identical
- scalable memory

mesh	V	E
brack2	62631	366559
mesh100	103081	200976
fe-ocean	143437	409593
dime20	224843	336024

$P = 16$	$\lvert E_c \rvert$	$t_s(s)$	$t_p(s)$	speed up
brack2	13540	9.45	3.66	2.58
mesh100	4596	8.54	2.22	3.85
fe-ocean	10943	13.87	3.26	4.25
dime20	1518	7.29	1.80	4.05

$P = 32$	$\|E_c\|$	$t_s(s)$	$t_p(s)$	speed up
brack2	20104	13.35	3.66	3.65
mesh100	7414	10.32	1.92	5.38
fe-ocean	16344	18.36	3.17	5.79
dime20	2402	8.59	1.22	7.04

$P = 64$	$\|E_c\|$	$t_s(s)$	$t_p(s)$	speed up
brack2	28983	17.86	4.19	4.26
mesh100	10923	12.49	1.46	8.55
fe-ocean	25737	25.38	2.52	10.07
dime20	3796	10.51	0.96	10.95

Dynamic Results

Averages over a series of 9 meshes
$N = 31172 \to 224833$.

JOSTLE-MS (multilevel static)
existing partition ignored
JOSTLE-MD (multilevel dynamic)
existing partition reused
JOSTLE-D (dynamic)
existing partition reused, no graph reduction

$P = 16$	$\lvert E_c \rvert$	$t(s)$	migrated %
JOSTLE-D	866	0.51	0.62
JOSTLE-MD	802	2.52	4.08
JOSTLE-MS	879	3.81	93.96
METIS	913	4.83	94.36
MRSB	939	55.85	83.54
GREEDY	1816	0.77	81.62

$P = 32$	$\lvert E_c \rvert$	$t(s)$	migrated %
JOSTLE-D	1568	0.66	1.82
JOSTLE-MD	1472	2.84	7.49
JOSTLE-MS	1488	4.20	92.77
METIS	1543	4.91	95.94
MRSB	1577	71.42	90.01
GREEDY	2897	0.83	90.64

$P = 64$	$\lvert E_c \rvert$	$t(s)$	migrated %
JOSTLE-D	2537	0.89	4.00
JOSTLE-MD	2380	3.35	9.67
JOSTLE-MS	2417	4.98	99.00
METIS	2427	5.15	97.95
MRSB	2520	87.34	95.07
GREEDY	4300	1.00	94.42

Initial partition

For global optimisation, quality of final partition should be independent of initial partition.

Example initial partitions:

- cyclic (parallel input)
- random (parallel input)
- block (parallel input)
- geometric sort (parallel sorting)
- greedy (serial)

Results for different initial partitions

initial algorithm	$\lvert E_c \rvert_0$	$\lvert E_c \rvert$	$t(s)$
random	28083	1895	1.50
block	11590	1896	1.22
geosort	1854	1864	0.85
greedy	2191	1888	0.79

whitaker3: $N = 9800$, $E = 28989$, $P = 32$

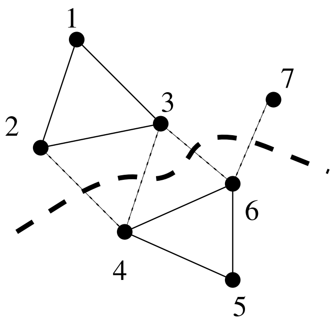
initial algorithm	$\lvert E_c \rvert_0$	$\lvert E_c \rvert$	$t(s)$
random	45174	2958	2.94
block	10653	3014	1.85
geosort	5416	2900	1.98
greedy	4007	2891	1.52

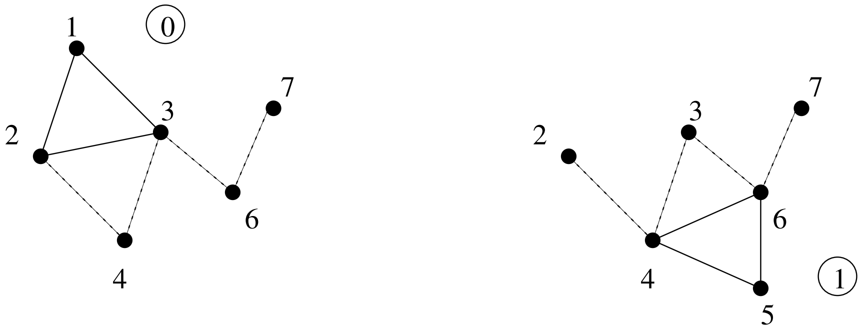
4elt: $N = 15606$, $E = 45878$, $P = 64$

Using Parallel JOSTLE at Stuttgart

```
pjostle_init_(&nprocs, &pid);
jostle_env_("format = node");
pjostle_(&nnodes, &offset, &core, &halo,
 index, degree, node_wt, partition,
 &local_nedges, edges, edge_wt,
 machine, user_machine, &output_level,
 &dimension, coords);
```

Example:





Node based data format:

The values of the arrays for processor 0 are

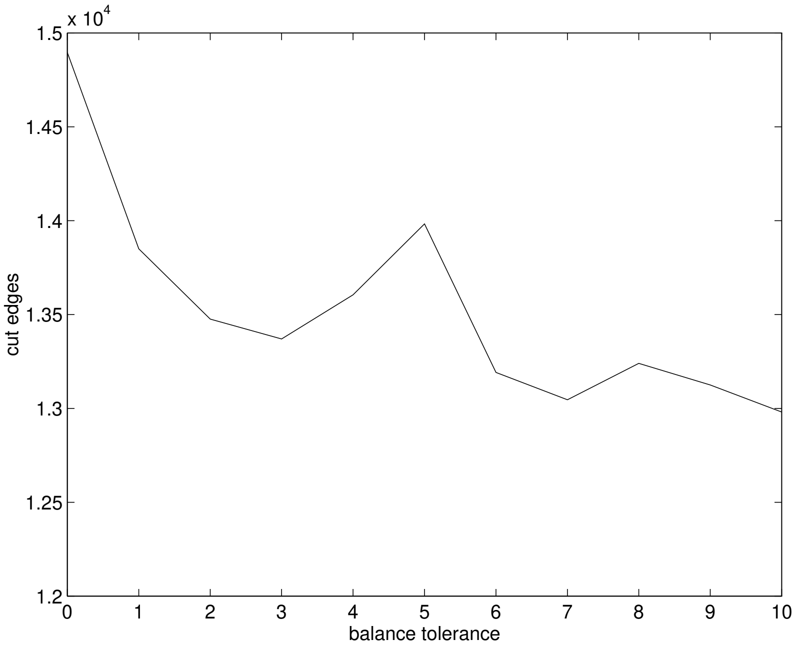
`index`	`[1,2,3,7,4,6]`
`degree`	`[2,3,4,1,4,4]`
`node_wt`	`(int*) NULL`
	OR
`node_wt`	`[1,1,4,1,1,1]`
	node 3 has weight 4
`partition`	`[-,-,-,-,1,1]`
`edges`	`[2,3,1,3,4,1,2,4,6,6]`
`edge_wt`	`(int*) NULL`
	OR
`edge_wt`	`[1,1,1,3,1,1,3,1,1,1]`
	(2,3) has weight 3
`coords`	`(double*) NULL`

Customisation: balance tolerance

```
jostle_env_("balance tolerance = 5");
```

allows an imbalance of 1.05 (5%)
optimal weight $W := \lceil |N|/P \rceil$
imbalance $= \max |S_p|/W$
default is 1.03

Tradeoff: imbalance $\leftrightarrow$ partition quality



Customisation: gather threshold

```
jostle_env_("gather threshold = 30");
```

create global graph when less than
30 nodes per processor
default is 20

Tradeoff: computation $\leftrightarrow$ communication

- fast computation
  increase gather threshold
- fast communication
  decrease gather threshold

Dynamic repartitioning

JOSTLE-MD (multilevel dynamic)

```
jostle_env_("matching = local");
jostle_env_("threshold = 20");
```

JOSTLE-D (dynamic)

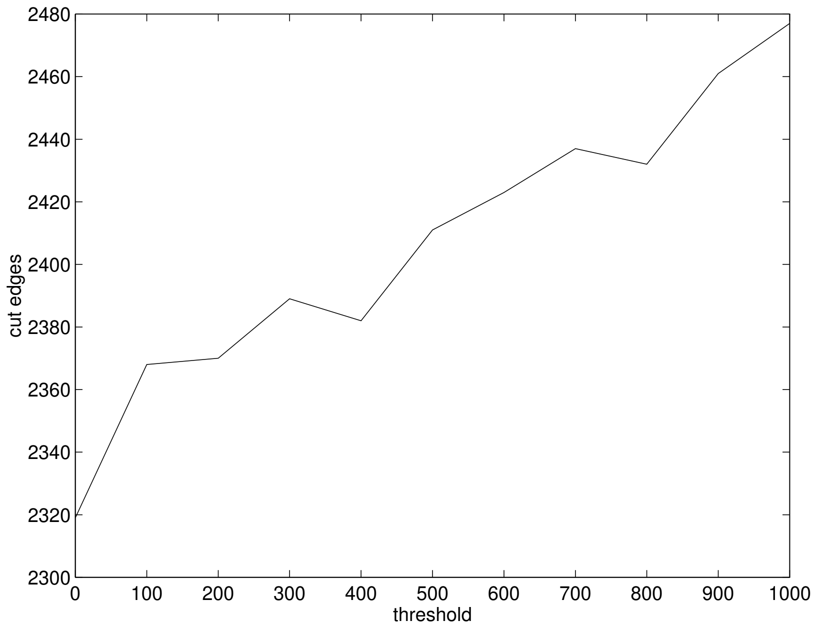
```
jostle_env_("reduction = off");
```

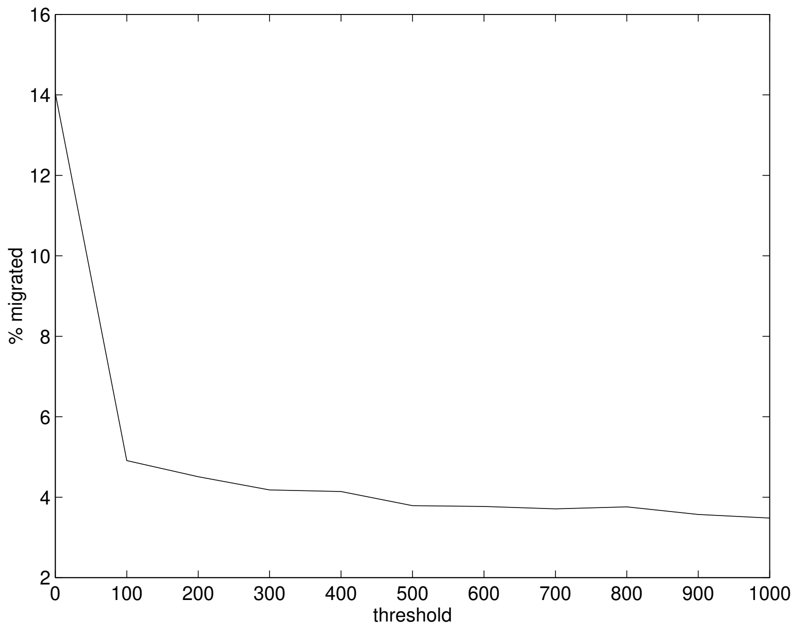
OR:

```
jostle_env_("matching = local");
jostle_env_("threshold = N");
```

Tradeoff: data migrated $\leftrightarrow$ partition quality

Dynamic repartitioning





Future Directions

- Unbalanced partitions
- Machine mapping
- Imbalance monitor
- Generic optimisation
  $\Rightarrow$ e.g. Subdomain Aspect Ratio