A Short Course in Parallel Scientific Computing

Technical University of Budapest

Graham Horton, University of Erlangen-Nürnberg

Talk 6 : Load Balancing

Contents

Motivation

Partitioning a problem for parallel execution

Each processor should receive an equal load

Parallel computation time $T(p)$ is the time for the slowest processor

$$T(p) = \max_i \; T_{p_i}$$

The efficiency is strongly dependent on a good load balance

$$E = \frac{T(1)}{pT(p)}$$

Example: 64 work units (e.g. grid points, grid columns)

on 8 processors:

$$E = \frac{64}{8 * 8} = 100\%$$

on 9 processors:

$$E = \frac{64}{8 * 9} \approx 89\%$$

64 work units cannot be distributed equally on 9 processors

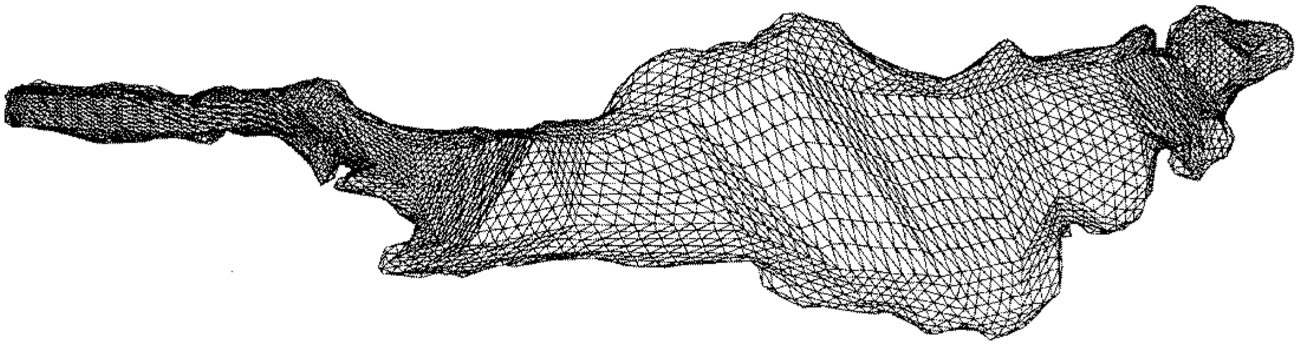
Motivation

LB is sometimes very easy (for example our programming examples)

It can be done by hand

Unfortunately this is not always the case

Example: Solution of a PDE with an unstructured grid:



We must find an algorithm for the load balancing of such a problem

Static and Dynamic Load Balancing

There are two classes of load balancing algorithms:

- Static LB : One-time assignment of work to processors
- Dynamic LB : Repeated reassignment of work units

Static LB is used when

- The load is known before the parallel program starts and
- The load does not change during the computation

Dynamic LB is used when

- The load is not known before the parallel program starts or
- The load does changes during the computation

Examples:

- (SLB) Iterative solution methods on unchanging grids
- (SLB) Matrix multiplication
- (DLB) Fractal computation (Mandelbrot set)
- (DLB) Branch and Bound methods

Static and dynamic Load Balancing

Static LBM have the properties

- No a priori information is needed
- Often difficult to respect additional conditions
- Are often run as a sequential preprocessing step
- The problem with this: memory!
- Are only run once: can afford to be expensive

Dynamic LBM have the properties

- Must respect existing load distribution
- Small changes in load must result in small redistributions
- Are called often: must be cheap
- Must be parallel
- Should make do with little synchronisation or global information

Static and dynamic Load Balancing

Example for necessity for dynamic LB:

Adaptive solution of a PDE

Approach:

1. Solve the PDE on the given grid
2. Estimate discretization error
3. Refine grid correspondingly
4. Goto 1

Goal : optimal sampling of the solution function

Minimisation of the discretisation error with given number of grid points

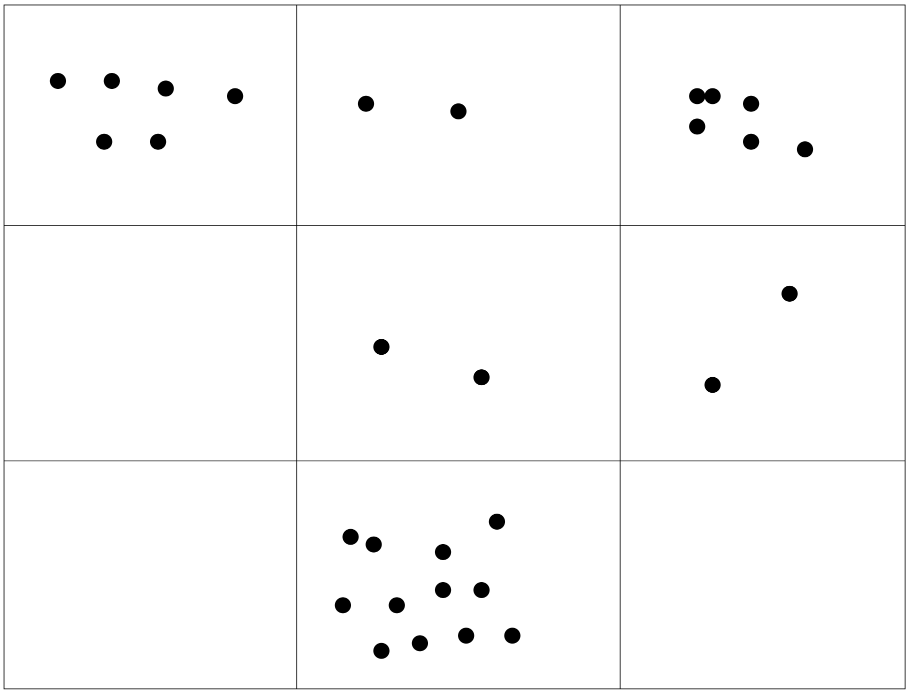
$\Rightarrow$ Need redistribution of grid points to processors

Static and dynamic Load Balancing

Example : Game of Life

Partition the "world" onto processors

Computation only takes place in the vicinity of living cells



Number and location of living cells changes constantly

$\Rightarrow$ Dynamic load balancing

Game of Life is very cheap $\Rightarrow$ LBM must be very fast

Properties of Load Balancing Methods

Classify LBM according to the following criteria:

- Dynamic / static
- Sequential / parallel execution
- Centralised / decentralised decisions
- Deterministic / non-deterministic

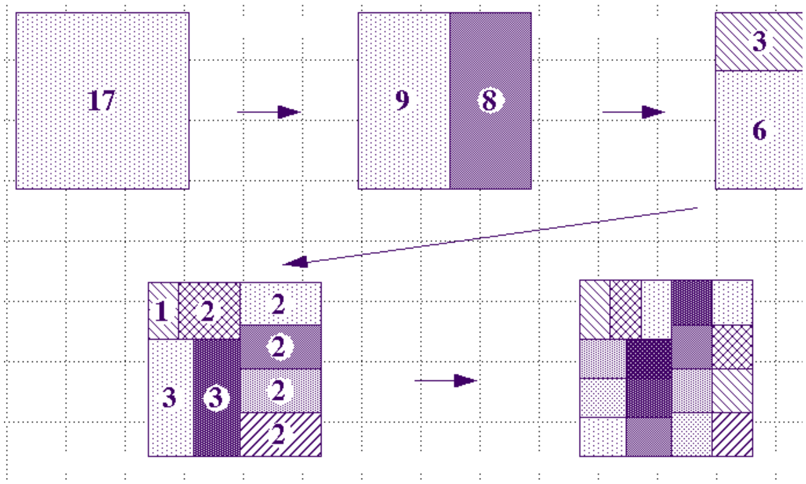
We must be able to

- Compute (estimate) the future load, or
- Measure the current load
- Move load between processors

The Recursive Bisection Method

Recursive Bisection methods are well-known static LBMs

Utilise the "Divide and Conquer" principle



Assume initially everything is on one processor

The load is recursively bisected along with the processor set

The size of the subproblems corresponds to the number of processors

(In a grid) The direction of subdivisions alternates

Problem: Can result in irregular problem topologies

In this example also consideration of communication overhead

In general: Communication minimisation is part of the load balancing

The Recursive Bisection Method

Denote P: processor set, T: subgrid

The general algorithm has the structure

```
bisection(P, T)
if (|P| = 1)
    assign T to proc. P
else
    divide P into P1 and P2
    divide T correspondingly into T1 and T2
    bisection(P1,T1)
    bisection(P2,T2)
```

We make no assumptions $\Rightarrow$ static LBM

Partial parallelism is present

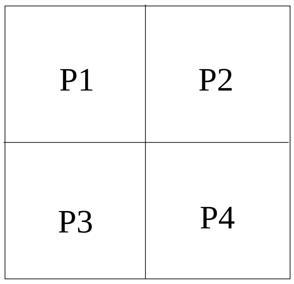
Not appropriate for dynamic LBM, bcause:

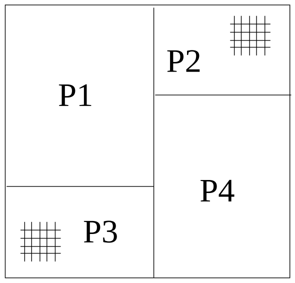
A small load change can lead to large redistribution of load

Example: next page

The Recursive Bisection Method

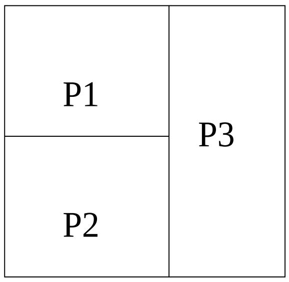
Example for change in problem topology due to small load change:

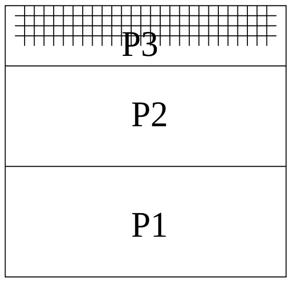




Example for large redistribution caused by small load change:

Lokale Gitterverfeinerung an einer Seite





Load balanced solutions with minimal communcation overhead

Inappropriate for dynamic LBM, because complete redistribution is necessary

The Diffusion Method

One simple LBM is the *Diffusion method*

Analogy to physical diffusion in solids

Idea: Each processor equalises load with its neighbours

Define: l_i is the current load in processor p_i

The general algorithm has the structure

```
while (not balanced)
     for all processors p_i
        for all nb processors p_j
           exchange l_i and l_j
           move 0.5 * (l_j - l_i) work units from p_i to p_j
```

Load balance will not be achieved in one step

$\rightarrow$ It is an iterative method (cf. Jacobi/ Gauß-Seidel)

Load exchange is based on local decisions

balanced : $|l_i - l_j| \leq 1, \quad \forall i, j \quad nb(p_i, p_j)$ local!

The Diffusion Method

Example : 6 processors $p_0, \ldots p_5$

Topology : Chain (linear array)

Initial condition:

$$Load(p_i) = \begin{cases} 30 & i = 0 \\ 0 & otherwise \end{cases}$$

Observe loads after each diffusion step $\Rightarrow$ next slide

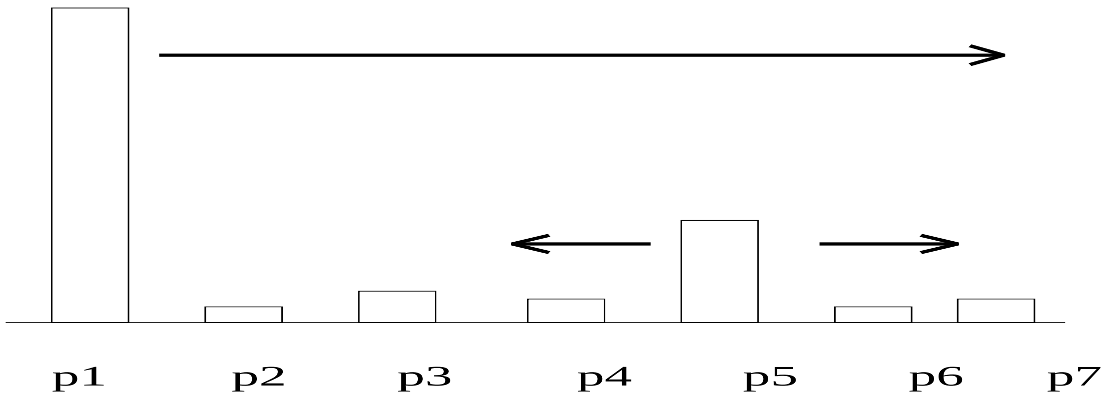
Result:

- Large number of steps
- Inaccurate load balance

For large p and initial load imbalance holds: $\#Iter = \mathcal{O}(p^2)$

It is not true that local balance $\Rightarrow$ global balance!

Also: Movement of load backwards and forwards is possible:



The Diffusion Method

Trace of the diffusion method:

Phase	*Load* (p_i)					
0	30	0	0	0	0	0
1	15	15	0	0	0	0
2	15	8	7	0	0	0
3	12	11	4	3	0	0
4	12	8	7	2	1	0
5	10	10	5	4	1	0
6	10	8	7	3	2	0
7	9	9	5	5	1	1
8	9	7	7	3	3	1
9	8	8	5	5	2	2
10	8	7	6	4	3	2
11	8	7	5	5	3	2
12	8	6	6	4	4	2
13	7	7	5	5	3	3
14	7	6	6	4	4	3
15	7	6	5	5	4	3

Multi-Level Method

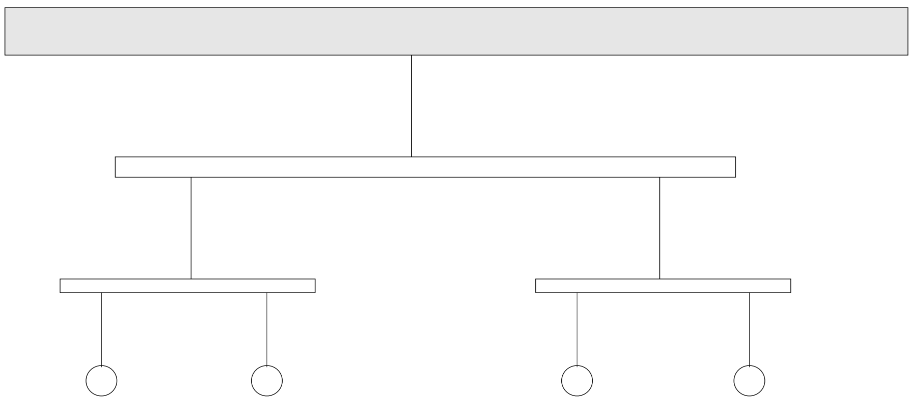
The diffusion method has the disadvantages:

- inaccurate
- slow

These result from the purely local nature of the decisions

In order to avoid these disadvantages, extend to *Multi-Level* method

Motivation: Mobile



Assumption: Hanging point of each beam can be moved

Idea: load exchange between groups of processors

Processor groups are arranged hierarchically

Multi-Level Method

Denote a processor set by P_i

The total load of processor set P_i is L_i (extra load)

The method is:

```
procedure balance(P : set of subproblems)
begin
   if ||P|| = 1 then return
   bisect P into P1 and P2
   calculate L1, L2
   transfer ⌊(L2||P1|| − L1||P2||)/(||P1|| + ||P2||)⌋ work units from P1 to P2
   balance (P1)
   balance (P2)
end balance
```

N.B. all neighbouring processors in P_1, P_2 may exchange load

It must hold:

$$
\begin{aligned}
P_1 \cap P_2 &= \emptyset \\
P_1 \cup P_2 &= P \\
|\ \|P_1\| - \|P_2\|\ | &\leq 1
\end{aligned}
$$

Multi-Level Method

The M-L algorithm computes load balance in $\lceil ld(\|P\|) \rceil$ *steps*

Proof: Induction over recursion level:

Let $lb(P)$ *state that the set* P *be load-balanced*

$$lb(P) \Leftarrow lb(P_1) \wedge lb(P_2) \wedge (|L_1 - L_2| \leq 1) \quad .$$

1) Transfering $\left\lfloor \frac{L_2\|P_1\| - L_1\|P_2\|}{\|P_1\| + \|P_2\|} \right\rfloor$ *load units from* P_1 *to* P_2 *leads to*

$$\begin{aligned} L_1^{new} &= \left\lceil \frac{L_1\|P_1\| + L_2\|P_1\|}{\|P_1\| + \|P_2\|} \right\rceil \\ L_2^{new} &= \left\lfloor \frac{L_1\|P_2\| + L_2\|P_2\|}{\|P_1\| + \|P_2\|} \right\rfloor \quad , \end{aligned}$$

which results in $|L_1 - L_2| \leq 1$

2) $lb(P_1)$ *and* $lb(P_2)$ *are satisfied by recursion of* `balance`.

3a) Processor sets are halved in size $\Rightarrow \lceil \mathrm{ld}(\|P\|) \rceil$ *levels of recursion*

3b) Parallel execution within each recursion level

Comparison Diffusion / Multi-Level

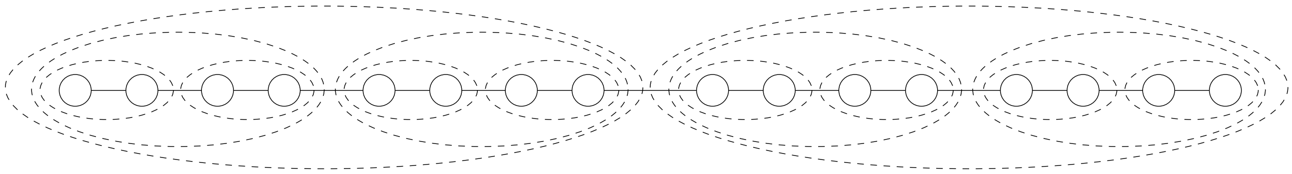
Test case: 16 processors

Initial state:

$$Load(p_i) = \begin{cases} 16 & i = 0 \\ 0 & otherwise \end{cases}$$

Topology: linear array

Processor sets and load balancing steps:



Movement of load backwards and forwards is not possible

Comparison Diffusion / Multi-Level

$Phase$	L															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16															
1	8	8														
2	8	4	4													
3	6	6	2	2												
4	6	4	4	1	1											
5	5	5	3	2	1											
6	5	4	4	2	1											
7	5	4	3	3	1											
8	5	4	3	2	2											
9	5	4	3	2	1	1										

$\#$ of steps grows quadratically with p and result is not exact

$Phase$	L															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16															
1	16							-8	8							
2	16			-12	12			-8	8			-4	4			
3	16	-14	14	-12	12	-10	10	-8	8	-6	6	-4	4	-2	2	
4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

$\#$ of steps grows logarithmically with p and result is exact