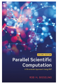
Laplacian Matrices

Section 4.9 of Parallel Scientific Computation, 2nd edition

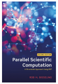
Rob H. Bisseling

Utrecht University



Physical domain

- In many applications, a physical domain exists that can be distributed naturally by assigning a subdomain to every processor.
- Communication is only needed for exchanging information across the subdomain boundaries.
- Often, the domain is structured as a multidimensional rectangular grid, where grid points interact only with a set of immediate neighbours.
- In the 2D case, these could be the neighbours to the north, east, south, and west.
- Example: the heat equation, where the value at a grid point represents the temperature at the corresponding location.



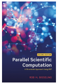
2D Laplacian operator for a $k \times k$ grid

Compute

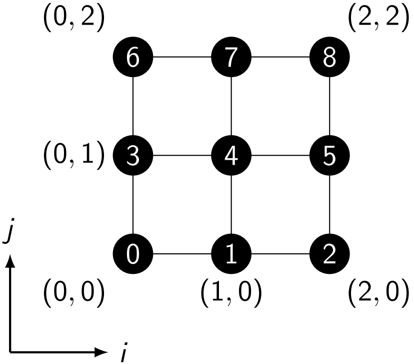
$$\Delta_{i,j} = x_{i-1,j} + x_{i+1,j} + x_{i,j+1} + x_{i,j-1} - 4x_{i,j}, \quad \text{for } 0 \leq i, j < k,$$

where $x_{i,j}$ denotes the temperature at grid point (i, j).

- By convention, $x_{i,j} = 0$ outside the grid.
- $x_{i+1,j} - x_{i,j}$ approximates the derivative of the temperature in the i-direction.
- $(x_{i+1,j} - x_{i,j}) - (x_{i,j} - x_{i-1,j}) = x_{i-1,j} + x_{i+1,j} - 2x_{i,j}$ approximates the second derivative.

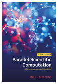


Relation grid–vector



- A 3×3 grid, which corresponds to a vector of length 9.
- For each grid point (i, j), the index $i + 3j$ of the corresponding vector component is shown.
- More in general,

$$v_{i+jk} \equiv x_{i,j}, \quad u_{i+jk} \equiv \Delta_{i,j}, \quad \text{for } 0 \leq i, j < k.$$



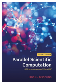
Relation operator–matrix

$$A = \begin{bmatrix} -4 & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & -4 & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & -4 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot & -4 & 1 & \cdot & 1 & \cdot & \cdot \\ \cdot & 1 & \cdot & 1 & -4 & 1 & \cdot & 1 & \cdot \\ \cdot & \cdot & 1 & \cdot & 1 & -4 & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & -4 & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & -4 & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & -4 \end{bmatrix}$$

$$\mathbf{u} = A\mathbf{v} \quad \Longleftrightarrow$$

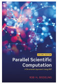
$$\Delta_{i,j} = x_{i-1,j} + x_{i+1,j} + x_{i,j+1} + x_{i,j-1} - 4x_{i,j}, \quad \text{for } 0 \leq i,j < k.$$

5 flops!



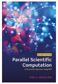
Domain view vs. matrix view

- In general, it is best to view the Laplacian as an operator on the physical domain.
- This domain view has the advantage that it naturally leads to the use of a regular data structure.
- Occasionally, however, it may be beneficial to view the Laplacian as a matrix, so that we can apply our knowledge about sparse matrix–vector multiplication.

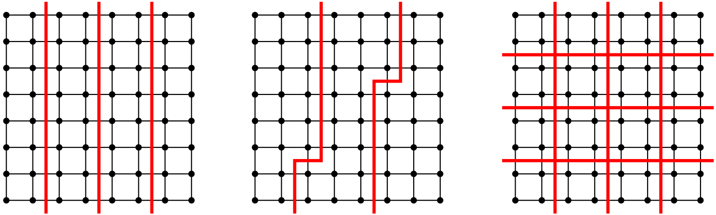


Find a domain distribution

- Here, we adopt the domain view, so that we must assign each grid point to a processor.
- We assign the values $\Delta_{i,j}$ and $x_{i,j}$ to the owner of grid point (i, j), which translates into $\mathrm{distr}(\mathbf{u}) = \mathrm{distr}(\mathbf{v})$.
- We use a row distribution for the matrix and assign row $i + jk$ to the same processor as vector components u_{i+jk} and v_{i+jk}, and hence grid point (i, j).
- The resulting parallel sparse matrix–vector multiplication has two supersteps: fanout and local matrix–vector multiplication.
- For our 2D grid, we decree the computation time to be:
  - for an interior point 5 flops;
  - for a border point 4 flops;
  - for a corner point 3 flops.



Distribution into strips and blocks

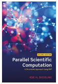


- Left: distribution into strips with long Norwegian/Chilean borders,

$$T_{\text{comm, strips}} = 2kg \qquad (\text{for } p > 2).$$

- Middle: boundary corrections improve the load balance.
- Right: distribution into square blocks with shorter borders,

$$T_{\text{comm, squares}} = \frac{4k}{\sqrt{p}} g \qquad (\text{for } p > 4).$$

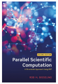


Surface-to-volume ratio

- The communication-to-computation ratio for square blocks is

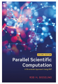
$$\frac{T_{\text{comm, squares}}}{T_{\text{comp, squares}}} = \frac{4k/\sqrt{p}}{5k^2/p} g = \frac{4\sqrt{p}}{5k} g.$$

- This ratio is also called the surface-to-volume ratio, because in 3D:
  - the surface of a domain represents the communication with other processors;
  - the volume represents the amount of computation of a processor.

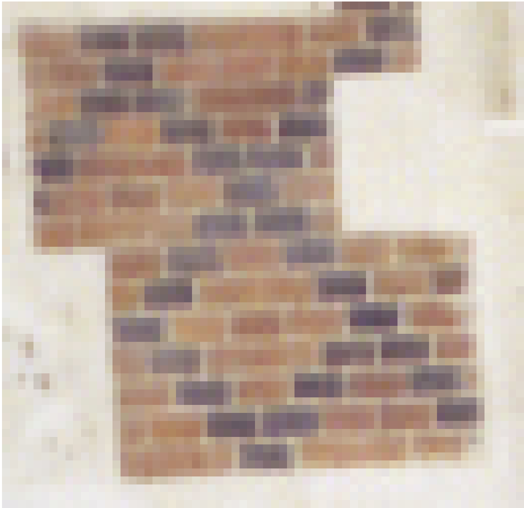


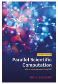
What do we do at scientific workshops?

- Participants of HLPP 2001, International Workshop on High-Level Parallel Programming, Orléans, France, June 2001, studying Château de Blois during an excursion.
- HLPP is held annually and it attracks many researchers from the BSP community.

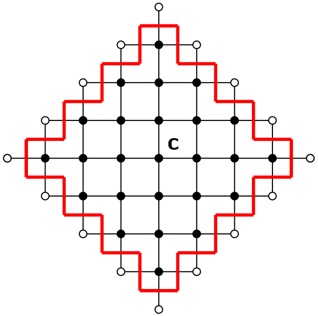


The high-level, low-resolution object of our study





Blocks are nice, diamonds ...



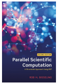
$r = 3$

- Digital diamond, or closed l_1-sphere, defined by

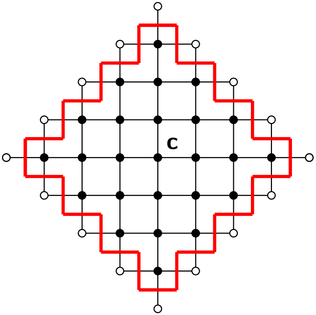
$$B_r(c_0, c_1) = \{(i,j) \in \mathbf{Z}^2 : \ |i - c_0| + |j - c_1| \le r\},$$

for integer radius $r \ge 0$ and centre $\mathbf{c} = (c_0, c_1) \in \mathbf{Z}^2$.

- $B_r(\mathbf{c})$ is the set of points with Manhattan distance $\le r$ to the central point $\mathbf{c}$.



Points of a diamond

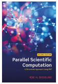


$r = 3$

- The number of points of $B_r(\mathbf{c})$ is

$$1 + 3 + 5 + \cdots + (2r-1) + (2r+1) + (2r-1) + \cdots + 1$$
$$= 2\sum_{k=0}^{r-1}(2k+1) + (2r+1) = 4\sum_{k=0}^{r-1} k + 4r + 1$$
$$= 2(r-1)r + 4r + 1 = 2r^2 + 2r + 1.$$

- The number of neighbouring points is $4r + 4$.



Diamonds are forever

- Assume that the diamond has its fair share of the grid points,

$$2r^2 + 2r + 1 = \frac{k^2}{p}.$$

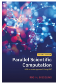
- Therefore, $2r^2 \approx \frac{k^2}{p}$ for large r, and hence

$$r \approx \frac{k}{\sqrt{2p}}.$$

- Just on the basis of $4r + 4$ receive operations, we have

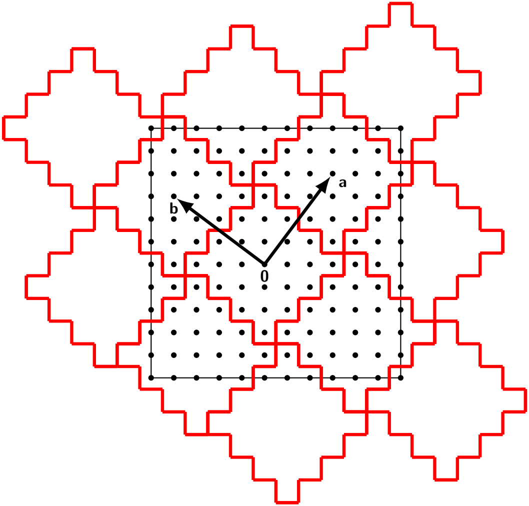
$$\frac{T_{\text{comm, diamonds}}}{T_{\text{comp, diamonds}}} = \frac{4r+4}{5(2r^2+2r+1)} g \approx \frac{2}{5r} g \approx \frac{2\sqrt{2p}}{5k} g.$$

- Compare with value $\frac{4\sqrt{p}}{5k} g$ for square blocks: factor $\sqrt{2}$ less.
- This gain is caused by reuse of data: each grid-point value sent is used twice.



Alhambra: tile the whole space

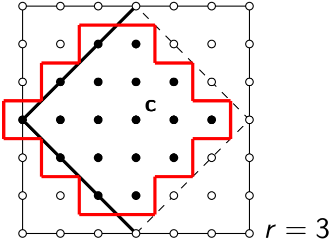
Tile the whole sky with diamonds



$r = 3$

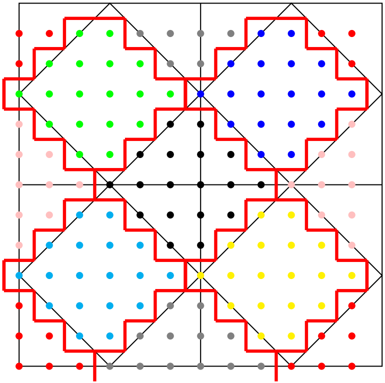
- Diamond centres at $\mathbf{c} = \lambda\mathbf{a} + \mu\mathbf{b}$, $\lambda, \mu \in \mathbf{Z}$, where $\mathbf{a} = (r, r+1)$ and $\mathbf{b} = (-r-1, r)$.
- This works well for an infinite grid.

Practical method for finite grids

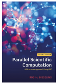


- Discard one layer of points from the north-eastern and south-eastern border of the diamond.
- For $r = 3$, the number of points decreases from 25 to 18.

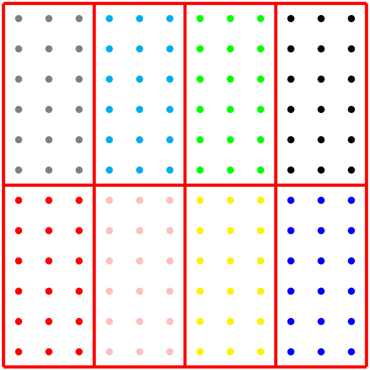
12×12 computational grid: periodic partitioning for $p = 8$



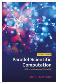
- Total computation: 672 flops. Avg 84. Max 90.
- Total communication: 104 values. Avg 13. Max 14.
- Total cost is $90 + 14g + 2l = 330$ for $g = 10$, $l = 50$.



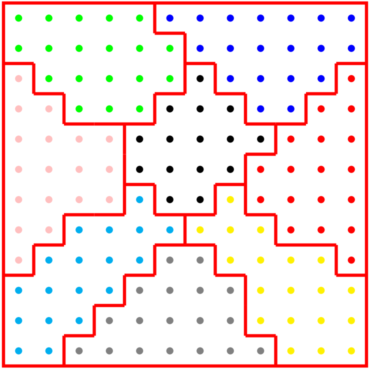
12×12 computational grid: block partitioning for $p = 8$



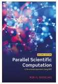
- Total computation: 672 flops. Avg 84. Max 87.
- Total communication: 96 values. Avg 12. Max 15.
- Total cost is $87 + 15g + 2l = 337$ for $g = 10$, $l = 50$.



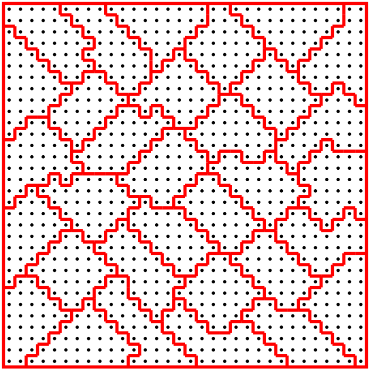
12×12 computational grid: Mondriaan partitioning



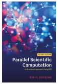
- Total computation: 672 flops. Avg 84. Max 89. ($\epsilon = 6\%$.)
- Total communication: 83 values. Avg 10.375. Max 14.
- Total cost is $89 + 14g + 2l = 329$ for $g = 10$, $l = 50$.
- Challenge: find a better solution by hand using ideas from both solutions shown.
- Lowest known cost (Bas den Heijer 2006): 299.



32×32 computational grid: Mondriaan partitioning



- Partitioning for $p = 32$ with $\epsilon = 6\%$.
- The total BSP cost is $165 + 23g + 2l$.
- Note the many diamond-like shapes, automatically discovered by Mondriaan.

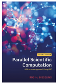


Three dimensions

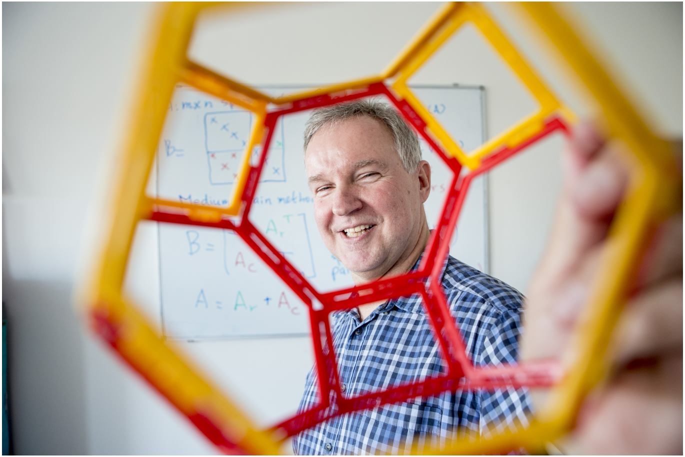
- In 3D, if a processor has a cubic block of $N = \frac{k^3}{p}$ points, the number of boundary points is about

$$6\left(\frac{k}{p^{1/3}}\right)^2 = \frac{6k^2}{p^{2/3}} = 6N^{2/3}.$$

- If a processor has a $10 \times 10 \times 10$ block, 488 points are on the boundary. About half!
- In 2D, the number of boundary points is only $4N^{1/2}$.
- Thus, communication is more important in 3D.
- A detailed analysis based on the surface-to-volume ratio of a 3D digital diamond shows that we can aim for a reduction by a factor $6^{1/3} \approx 1.82$ in communication cost.

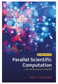


My truncated octahedron

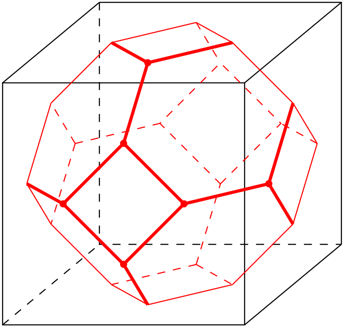


Photographer: Ivar Pel

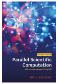
- We can tile 3D space with copies of this object.



Basic cell for 3D



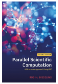
- Basic cell: grid points in a truncated octahedron.
- For load balancing, take care with the boundaries.
- What You See, Is What You Get (WYSIWYG):
  4 hexagons and 3 squares visible at the front are included.
  Also 12 edges, 6 vertices.
- Gain factor of 1.68 achieved for $p = 2q^3$.



Comparing communication costs for 3 distribution methods

Grid	p	Rect.	Diam.	Mondriaan	
		h	h	h	V/p
$4\,096 \times 4\,096$	8	5 120	4 098	5 587	4 106
	16	4 096	4 096	4 306	3 100
	32	3 072	2 050	3 303	2 468
	64	2 048	2 048	2 383	1 797
	128	1 536	1 026	1 728	1 313
$256 \times 256 \times 256$	16	49 152	37 250	50 676	38 474
	128	16 384	9 410	16 568	12 312

- Communication cost (in g) for a Laplacian operation on a 2D or 3D grid with 2^{24} grid points..
- Mondriaan version 4.2 was run in row-distribution mode with $\epsilon = 6\%$.
- In 2D, diamonds are better than blocks by a factor 1.50 for $p = 32, 128$. In 3D, by a factor 1.74 for $p = 128$.



Summary

- Communication can be reduced tremendously by using knowledge of the physical domain.
- To achieve a good distribution with a low surface-to-volume ratio, all dimensions must be cut. In 2D, this gives square blocks. In 3D, cubes.
- In 2D, an even better method is to use a digital diamond with some boundaries removed. This basic cell can be used to tile a rectangular domain. The best performance is obtained for $p = 2q^2$.
- In 3D, the best method is to use a truncated octahedron with WYSIWYG tie-breaking at the boundaries. The best performance is obtained for $p = 2q^3$.