a) The application being solved here is the graph coloring problem.

The Wikipedia definition of graph problem is as follows:

In graph theory, graph coloring is a special case of graph labeling; it is an assignment of labels traditionally called "colors" to elements of a graph subject to certain constraints. In its simplest form, it is a way of coloring the vertices of a graph such that no two adjacent vertices share the same color; this is called a vertex coloring. Similarly, an edge coloring assigns a color to each edge so that no two adjacent edges share the same color.

In this example we will consider vertex coloring.

b) The problem requires large/fast computation because the total number of graph nodes/vertices can reach up to billions for real world system. For ex: Webgraph, Twitter, facebook etc.

Just a straight-forward traversal would take hours to run.

c) The idea to solve graph coloring problem goes this way:

Every node is assigned a priority. A node can color itself if and if all its neighbor nodes with higher priority have colored themselves. Once a node colors itself it sends notification to all its neighbor that it has colored. Notice here that all those nodes which are about to color themselves can concurrently themselves without the need to worry about any conflict. This makes the algorithm suitable for running on multicore machine.

There are two parts in the algorithm:

i) Priority assignment:
There are multiple ways to compute priority. One way is to randomly assign a value to every node. This can also be done in parallel.
Another way would be to assign the priority based on total number of edges connected to that node. It is possible in this case that two nodes might end up getting same priority in which case the issue of who should color itself first can be resolved based on the node ID.

ii) Color assignment:
Every node that is activated can color itself. A node where in all its neighboring nodes of higher priority are colored is called an active node.
Every time a node colors itself, it sends notification to its neighboring nodes that it has colored itself and what color it has used. The neighboring nodes keeps a list of color that it should not use based on the notification it has received. Once the total number of color that should not be used reaches the total number of nodes that are of higher priority than the current node, the node is activated.

In the next step, all activated nodes color themselves in parallel.

d) Catalyst cluster under DOE/NNSA/LLNL, United States, which stands at 4th position in the top 500 computing machines in world, was used to perform the operation. Catalyst is a 324 node experimental data-intensive platform that has dual 12-core Intel Xeon E5-2695v2(2.4 GHz) processors, 128 GB of memory, and Intel 910 PCI-attached NAND Flash per node.

The machine has 1,572,864 cores with Rmax of 17173.2 and Rpeak of 20,132.7. Core specification is Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x.

e) The performance of the algorithm depends on the total number of edges for different nodes. For the algorithm to work well it is necessary that at a given time there are as many active nodes as possible so that they are all ready for coloring themselves. This way every core will be made use. Since the entire algorithm runs in parallel, it is scalable.

Reference:

[1] Graph Colouring as a Challenge Problem for Dynamic Graph Processing on Distributed Systems by Scott Sallinen, Keita Iwabuchi, Suraj Poudel, Maya Gokhale, Matei Ripeanu, Roger Pearce