Synthesis of Distributed Arrays in Titanium

Amir Kamil

U.C. Berkeley

May 9, 2006

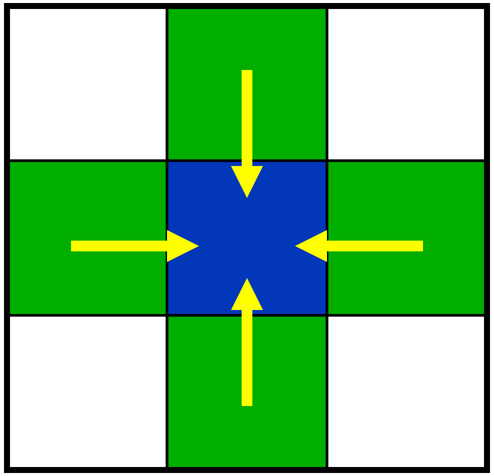
Background

- **Titanium is a single program, multiple data (SPMD) dialect of Java**
  - All threads execute the same program text
- **Designed for distributed machines**
- **Global address space – all threads can access all memory**
  - But much slower to access remote memory than local memory

Grids – The Abstract View

- **Grids used extensively in scientific codes**
- **Ideally, programmer specifies:**
  - Size of grid
  - Operations on each cell

```
grid[2d] g = new grid[[0,0] : [100,100]];
setup(g);
for (int i = 0; i < iterations; i++) {
  foreach (p in g.domain()) {
    g[p] = (g[p+[0,-1]] +
            g[p+[0,1]] +
            g[p+[1,0]] +
            g[p+[-1,0]]) / 4;
  }
}
```

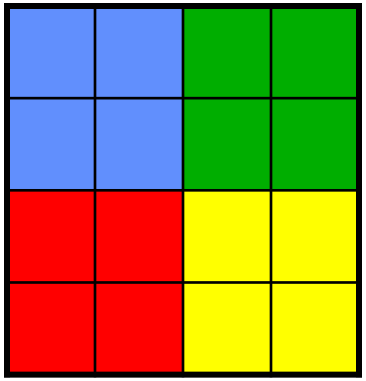


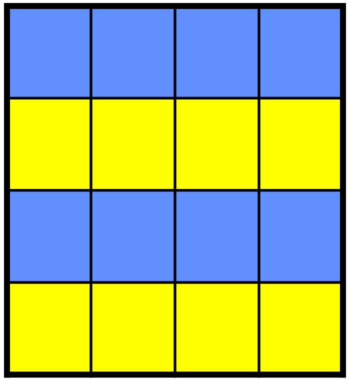
Grids – The Reality

- **Grids must be distributed across processors**
  - Global accesses are slow, local accesses are fast
  - Load balancing is difficult
- **Some problems require multiple levels of refinement**
- **Access patterns must be tailored for problem and machine**

Grid Distribution

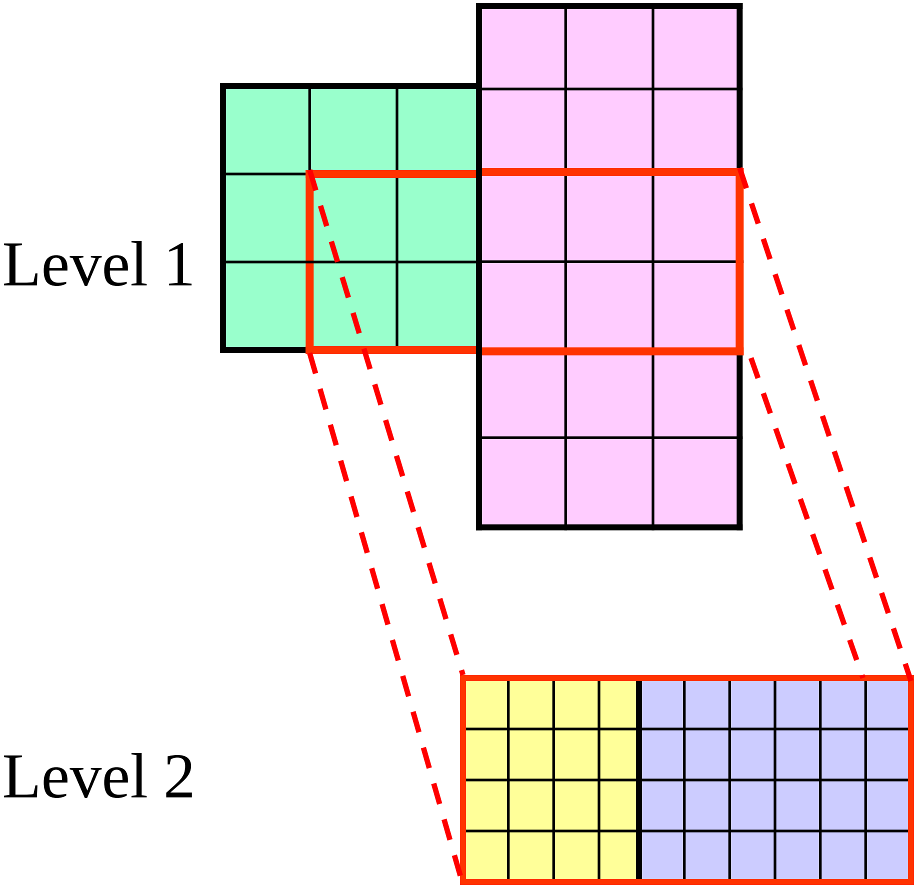
- **Regular partitioning**
  - Blocked, Cyclic distributions





- **Can also partition irregularly**
- **Ghost cells at boundaries used to cache data**

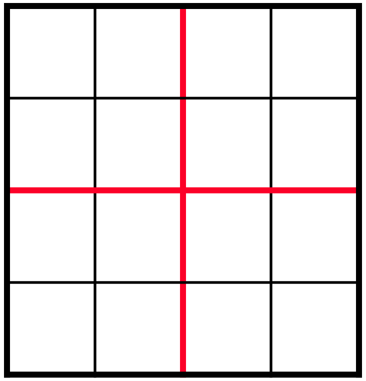
Multi-level Grids

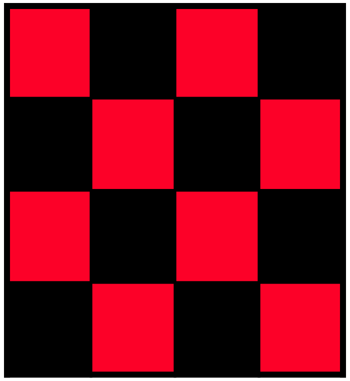


- Parts of the grid may require higher resolution
- Each level distributed separately
- Lower levels are refinements of upper levels
- Some notion of consistency between levels

Access Patterns

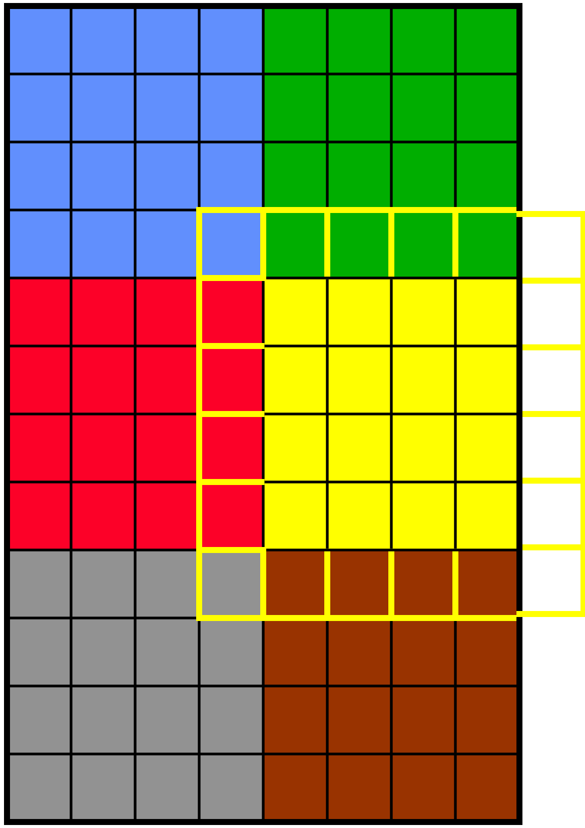
- **Different problems require different access patterns**
  - Data dependency
  - Cache effects
- **Examples:**
  - Blocked accesses (linear algebra)
  - Red/black (multigrid)





Problem #1 – Game of Life

- **2D grid**
  - Blocked in both dimensions
  - Ghost cells of width 1 at boundaries

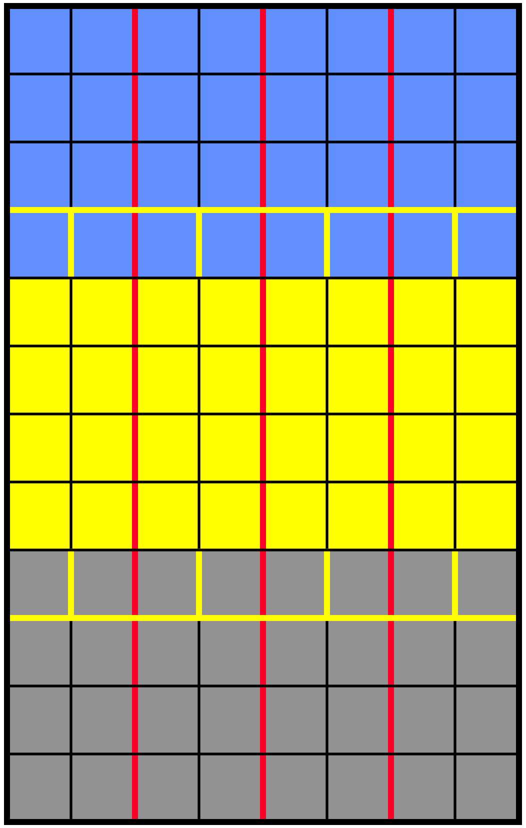


```
array data {
  dimension[2];
  distribution[BLOCKED(length[1] / 3),
               BLOCKED(3 * length[2] /
                       Ti.numProcs())];
  boundary[GHOST(1), GHOST(1)];
}
```

Problem #2 – Knapsack

- **2D grid**
  - Blocked in one dimension
  - Blocked access pattern in other dimension
  - Ghost cells of width 1 at boundaries

```
array data {
  dimension[2];
  distribution[BLOCKED(length[1] /
                      Ti.numProcs()),
               NONE];
  access[NONE, BLOCKED(2)];
  boundary[GHOST(1), NONE];
}
```



Grid Usage

- **Generated grids mostly used as if they're normal, global grids**
  - Array access (`[]`, `[]=`) to any cell supported
- **Ghost cells automatically updated by calling `synchronize()` method**
- **Methods provided to restrict access to local elements, specified pattern**
  - e.g. `myDomain()`, `myBlocks()`

Future Work

- **Optimize certain access patterns in compiler**
  - e.g. can remove owner computation when iterating over local domain

```
foreach (p in grid.myDomain())
 grid[p] = …
```

- **Add basic support for multiple levels of refinement**

Future Future Work

- **Add more distribution types**
  - e.g. Blocked-Cyclic
  - Irregular partitioning
- **Add load balancing**
- **Support irregular grids**
  - e.g. AMR
- **Add other boundary conditions**
  - e.g. shared cells
- **Improve compiler support by adding optimizations, analysis**