Massively Parallel Computer Architecture

Hiroshi Nakashima
(ACCMS, Kyoto University)

Contents of the Lecture
- Components & technologies in high-performance systems
  - high-performance microprocessors
  - shared memory systems
  - distributed memory systems
  - accelerators
- Methodologies of high-performance computing for;
  - explicit solver of diffusion equations
  - ( & matrix-matrix multiply, linear solvers, ...)
- Skills in high-performance programming with deep understanding of parallel systems

Course Management
- Course materials (Slides)
  - pptx/pdf files has been (or will be) distributed by graduate school office.
  - Paper-version handout is only for the first portion.
- Achievement evaluation
  - By exercise report.
  - Theme will be given on the last day.
  - Theme will be on high-performance programming (rather than "impression of lecture").

Solving Diffusion Equation (1/4)

for(t=0;t<tmax;t++) {
  for(y=0;y<ny;y++) for(x=0;x<nx;x++)
    un[y][x]=u[y][x]+
      (dt/(h*h))*(u[y][x+1]+u[y][x-1]+u[y+1][x]+u[y-1][x]-
      4*u[y][x]));
  tmp=un; un=u; u=tmp;
}

c.f. Similar Code (1)
Jacobi/red-black SOR solver of \( \nabla^2 \varphi = g \)

for(y=0;y<ny;y++) for(x=0;x<nx;x++)
  un[y][x]=a*(u[y][x-1]+u[y][x+1]+u[y-1][x]+u[y+1][x]-
  4*u[y][x]);
  for(odd=0;odd<2;odd++)
    for(y=0;y<ny;y++)
      for(x=odd^(y&1);x<nx;x+=2)
        u[y][x]=a*u[y][x]+b*(u[y][x-1]+u[y][x+1]+u[y-1][x]+u[y+1][x]-
                      4*u[y][x]));
Solving Diffusion Equation (1/4)

- c.f. Similar Code (2)

\[ \nabla \times \mathbf{E} = \frac{\partial \mathbf{B}}{\partial t} \n\]

```c
for(z=0;z<nz;z++)
  for(y=0;y<ny;y++)
    for(x=0;x<nx;x++)
      b[z][y][x].x +=
      e[z+1][y][x].y - e[z][y][x].y-
      e[z][y+1][x].z + e[z][y][x].z;
      b[z][y][x].y+=...
      b[z][y][x].z+=...
```

Parallelism & Locality (1/4)

- Principle of High-Performance = P + L
  - Parallelism
    - in: instructions/operations, innermost loops, outer loops, functions/procedures, programs, ...
    - by: hardware, compilers, programmers
  - Locality: Systems believe/expect that...
    - temporal: an event which happens now will likely happen again in near future; and
    - spatial: a series of temporally proximate events are likely proximate spatially; and thus codes against the belief/expectation should run very slowly.

Parallelism & Locality (2/4)

- Parallelism in DE-solver loop
  ```c
  for(t=0;t<tmax;t++) {
    for(y=0;y<ny;y++)
      for(x=0;x<nx;x++)
        un[y][x]=
        u[y][x]+(dt/(h*h))*(u[y][x+1]+u[y][x-1]+
        u[y+1][x]+u[y-1][x]-
        4*u[y][x]);
    tmp=un; un=u; u=tmp;
  }
  ```

Parallelism & Locality (3/4)

- u and un have loop-carry dependence
  - outermost loop cannot be parallelized
  ```c
  for(t=0;t<tmax;t++) {
    for(y=0;y<ny;y++)
      for(x=0;x<nx;x++)
        un[y][x]=
        u[y][x]+(dt/(h*h))*(u[y][x+1]+u[y][x-1]+
        u[y+1][x]+u[y-1][x]-
        4*u[y][x]);
    tmp=un; un=u; u=tmp;
  }
  ```

Parallelism & Locality (4/4)

- Spatial Locality in DE-solver loop
  ```c
  for(t=0;t<tmax;t++) {
    for(y=0;y<ny;y++)
      for(x=0;x<nx;x++)
        un[y][x]=
        u[y][x]+(dt/(h*h))*(u[y][x+1]+u[y][x-1]+
        u[y+1][x]+u[y-1][x]-
        4*u[y][x]);
    tmp=un; un=u; u=tmp;
  }
  ```