Massively Parallel Computer Architecture

Hiroshi Nakashima
(ACCMS, Kyoto University)

Introduction (1/2)

- 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**

Introduction (2/2)

- 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)

- Discretized & Approximated Solver of Initial/Boundary-Value Problem of 2-dimensional $\nabla^2\varphi = \frac{\partial\varphi}{\partial t}$

$$\varphi = u(x,y,t)$$

$$\nabla^2\varphi = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u(x+\Delta x,\, y,\, t) - 2u(x,\, y,\, t) + u(x-\Delta x,\, y,\, t)}{\Delta x^2}$$

$$\frac{\partial u}{\partial t} \approx \frac{u(x,\, y,\, t+\Delta t) - u(x,\, y,\, t)}{\Delta t}$$

$$u(x,y,t{+}\Delta t) = u(x,y,t) + \frac{\Delta t}{h^2}(u(x{+}h,y,t) + u(x{-}h,y,t) + u(x,y{+}h,t) + u(x,y{-}h,t) - 4u(x,y,t))$$

Solving Diffusion Equation (1/4)

$$u(x,y,t{+}\Delta t) = u(x,y,t) + \frac{\Delta t}{h^2}(u(x{+}h,y,t) + u(x{-}h,y,t) + u(x,y{+}h,t) + u(x,y{-}h,t) - 4u(x,y,t))$$

```
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;
}
```

Solving Diffusion Equation (1/4)

- 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]);

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]);
```

Solving Diffusion Equation (1/4)

- c.f. Similar Code (2)

$$\nabla \times \boldsymbol{E} = \frac{\partial \boldsymbol{B}}{\partial t}$$

```
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

```
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;
}
```

innermost loop

outer loop

instruction/operation-level parallelism

u and un have loop-carry dependence
➔outermost loop cannot be parallelized

Parallelism & Locality (3/4)

- Temporal Locality in DE-solver loop

```
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;
}
```

iterative execution of instructions in loop body
➔hit to instruction cache

continually accessed local scalar variables ➔ on-register

continual access to an array element
➔on-register?

not easily done in C
🡐the element can be updated by another assignment between two references
➔safely done in this case

continuous establishment of branch condition
➔basics of branch prediction

Parallelism & Locality (4/4)

- Spatial Locality in DE-solver loop

```
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;
}
```

thus x is inner

continuous inside/outside an iteration (⇔ Fortran)

instructions are ranked continuously