Pthreads Exercises

Karl Ljungkvist

16 February 2016

Example: Scalar product

$$\alpha = v \cdot w$$

Serial code:

```
int N=1000000;
double *v, *w;
double a=0;
for(i=0; i<N; ++i) {
  a += v[i]*w[i];
}
```

Idea:

- Parallelize for loop

Discussion:

- Problems?

Parallelization 1:

```
int N=1000000;
double *v, *w; double a=0;
struct arg_t {int i1,i2;};

void *dot(void *arg) {
  ...
  for(int i=i1; i<i2; ++i)
    a += v[i]*w[i];
  pthread_exit(NULL);
}
main() {
...
  int chunk = N/NUM_THREADS;
  for(t=0; t<NUM_THREADS; ++t) {
    arg[t].i1 = chunk*t;
    arg[t].i2 = chunk*(t+1);
    pthread_create(&thread[t],NULL,
                   dot,&arg[t]);
  }
  for(t=0; t<NUM_THREADS; ++t) // join
}
```

Example: Scalar product

Shared variable:

- Protection needed!
- Add mutex

Solution 2:

```
int N=1000000;
double *v, *w; double a=0;
pthread_mutex_t mutex;

void *dot(void *arg) {
  ...
  for(int i=i1; i<i2; ++i) {
    double b = v[i]*w[i];
    pthread_mutex_lock(&mutex);
    a += b;
    pthread_mutex_unlock(&mutex);
  }
  pthread_exit(NULL);
}
```

Discussion:

- Correct now?
- Other problems?

Example: Scalar product

Performance:

Serial	Parallel
0.0027 s	0.14 s

(Quad core processor)

Massive slowdown:

- Too much contention on lock

Improvement:

- Use local copy – reduce critical operations

New version:

```
void *dot(void *arg) {
  ...
  double aloc = 0;
  for(int i=i1; i<i2; ++i)
    aloc += v[i]*w[i];

  pthread_mutex_lock(&mutex);
  a += aloc;
  pthread_mutex_unlock(&mutex);
  pthread_exit(NULL);
}
```

New performance:

Serial	Parallel
0.0027 s	0.0015 s

Example: Numerical PDE Solver

Advection equation in 2D:

$$
\begin{aligned}
u_t + u_x + u_y &= F(t,x,y) && \text{at } 0 \le x \le 1, 0 \le y \le 1 && (PDE)\\
u(t,0,y) &= h_1(t,y) && \text{at } 0 \le y \le 1 && (BC_y)\\
u(t,x,0) &= h_2(t,x) && \text{at } 0 \le x \le 1 && (BC_x)\\
u(0,x,y) &= g(x,y) && && (IC)
\end{aligned}
$$

Solve with explicit Finit Difference Method (*Leapfrog*)

(In lab: `leapfrog.c`)

Example: Numerical PDE Solver

Core of the computations:

```
for k=2:Nt
  t=k*dt; Uold=U; U=Unew;
  for j=1:Ny-1
    for i=1,Nx-1
      x=i/Nx; y=j/Ny
      Unew(i,j)=Uold(i,j)+2*dt*(F(t,x,y)-
                (U(i+1,j)-U(i-1,j))/(2*dx)-
                (U(i,j+1)-U(i,j-1))/(2*dy))
    end for
  end for
end for
```

Discussion:

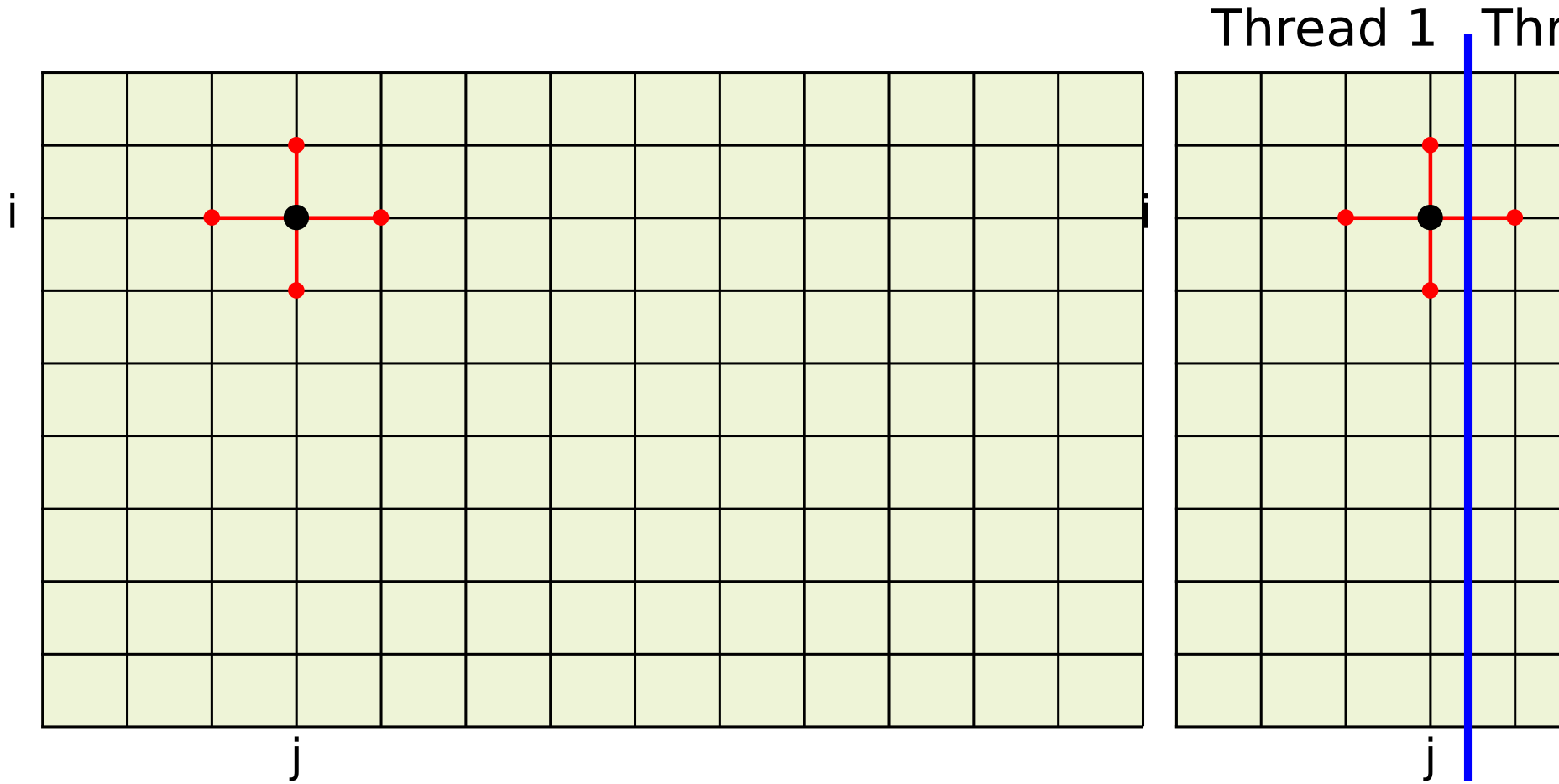
- Where is the parallelism?

Answer:

- Update of each element `Unew(i,j)` is perfectly parallel within the `k` loop.

Example: Numerical PDE Solver

Computation stencil:



Divide grid over the threads, parallelize over `j`.

Example: Numerical PDE Solver

Parallel thread tasks:

```
for k=2:Nt

  t=k*dt; Uold=U; U=Unew;
  for j=j1:j2
    for i=1,Nx-1
      x=i/Nx; y=j/Ny
      Unew(i,j)=Uold(i,j)+2*dt*(F(t,x,y)-
                (U(i+1,j)-U(i-1,j))/(2*dx)-
                (U(i,j+1)-U(i,j-1))/(2*dy))
    end for
  end for
end for
```

$\Rightarrow$ Perfectly parallel computations, `j` loop parallelized

Question:

- Will this work?
- Are there data dependencies?
- What happens when threads go to next iteration?

Example: Numerical PDE Solver

Dependency:

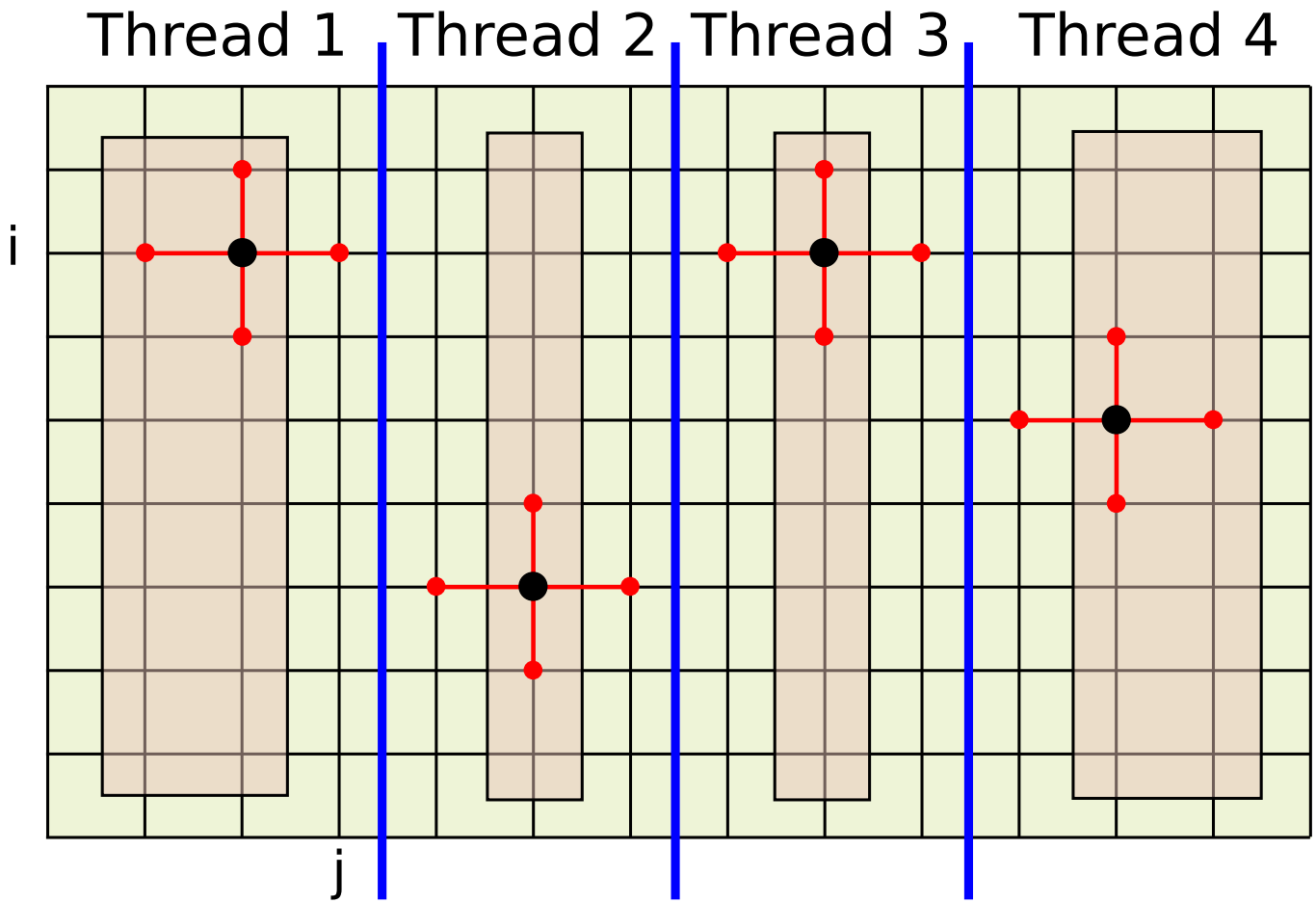
- Solution on step `k` depends on solution on step `k-1`
- Must wait for all threads to be done
- Synchronize in each time step – barrier

Correct version:

```
for k=2:Nt
  thread_barrier();
  t=k*dt; Uold=U; U=Unew;
  for j=j1:j2
    for i=1,Nx-1
      x=i/Nx; y=j/Ny
      Unew(i,j)=Uold(i,j)+2*dt*(F(t,x,y)-
                (U(i+1,j)-U(i-1,j))/(2*dx)-
                (U(i,j+1)-U(i,j-1))/(2*dy))
    end for
  end for
end for
```

Example: Numerical PDE Solver

Note: No need to have a *barrier*, just make sure that all threads are working with the same time step (iteration `k`). The inner points do not depend on other threads data, start computing on these points.



Example: Numerical PDE Solver

After computing on inner points, check if all threads have reached the same time step, i.e., started to compute on its inner points:

```
for k=2:Nt
  thread_barrier_start(); % thread starts a new step
  t=k*dt; Uold=U; U=Unew;
  for j=(j1+1):(j2-1)
    for i=1,Nx-1
      x=i/Nx; y=j/Ny
      Unew(i,j)=Uold(i,j)+2*dt*(F(t,x,y)-
                (U(i+1,j)-U(i-1,j))/(2*dx)-
                (U(i,j+1)-U(i,j-1))/(2*dy))
    end for
  end for
  thread_barrier_end(); % Wait until all threads
                        % have called the start barrier
  update Unew(:,j1) and Unew(:,j2)
end for
```

Example: Numerical PDE Solver

Barrier:

```
int ready = 0; /* shared */
int step = 0; /* shared */
int locstep = 0; /* per thread */
```

thread_barrier_start():

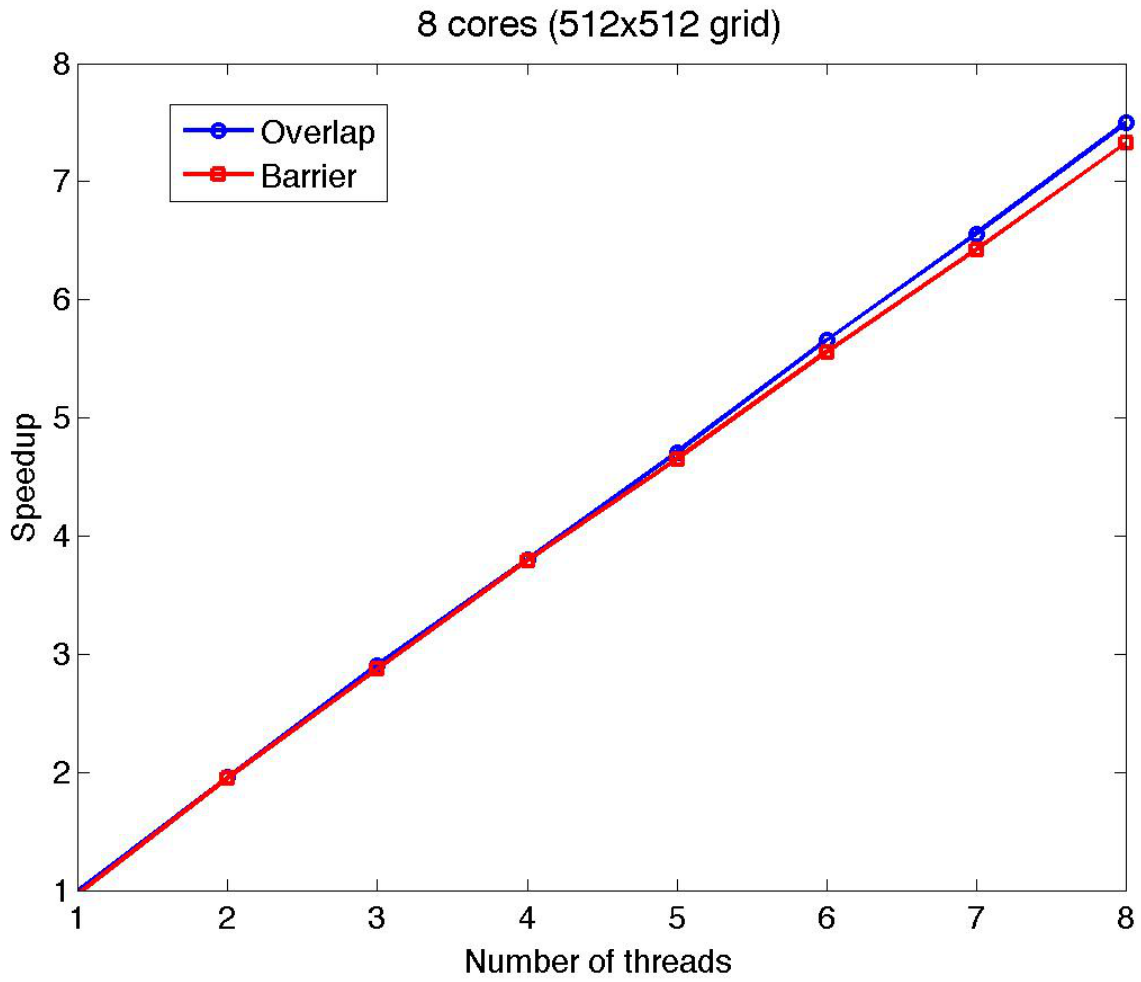
```
pthread_mutex_lock(&lock);
ready++;
locstep++;
if (ready==nthreads){
  ready=0;
  step++;
  pthread_cond_broadcast(&signal);
}
pthread_mutex_unlock(&lock);
```

thread_barrier_end():

```
pthread_mutex_lock(&lock);
while (locstep>step)
  pthread_cond_wait(&signal,&lock);
pthread_mutex_unlock(&lock);
```

Example: Numerical PDE Solver

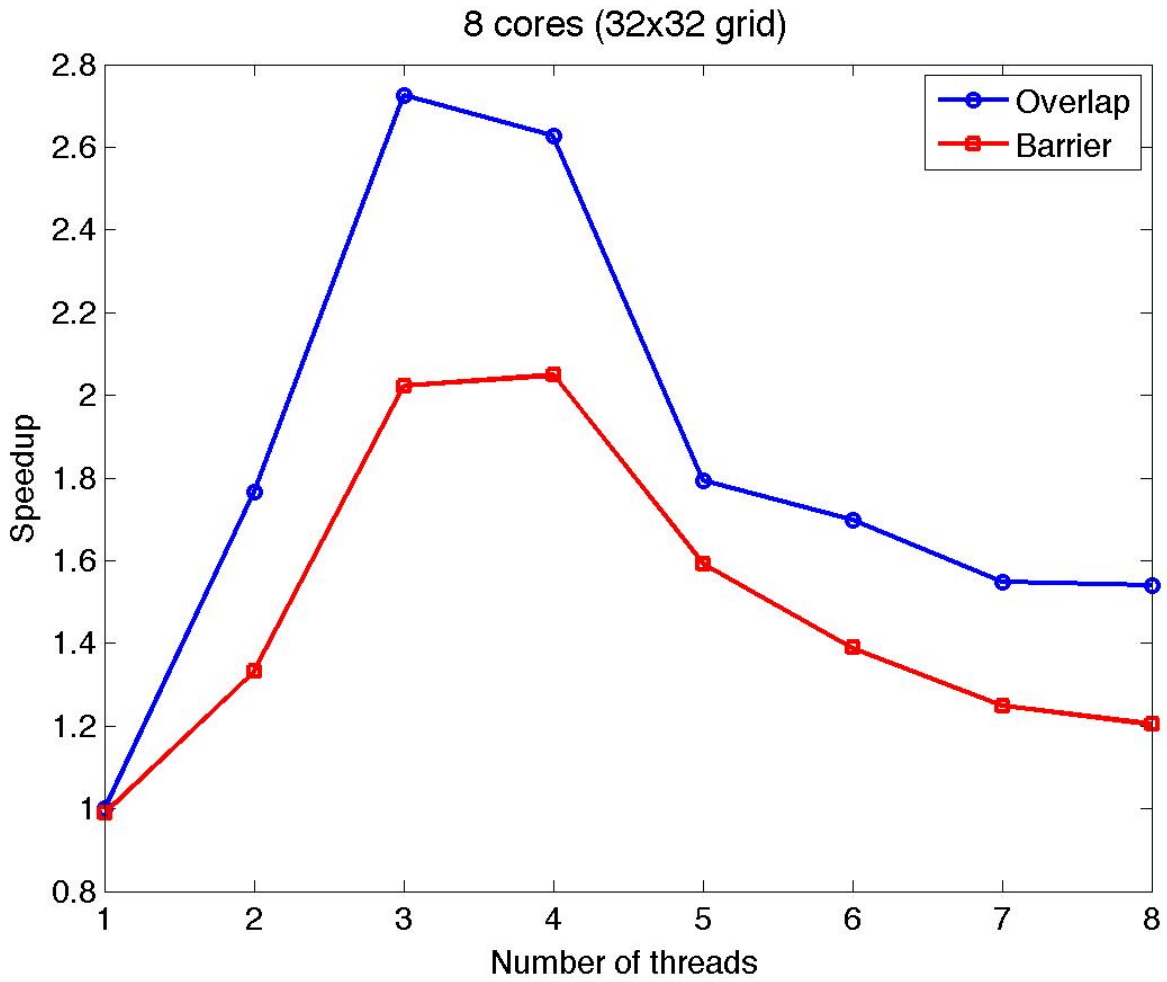
Performance on 8 cores (2x quadcore):



Example: Numerical PDE Solver

Small grid:

- Synchronization overhead becomes significant



Example: Numerical PDE Solver

Remark:

- Reduce thread swapping by letting master thread be peer in computations.
- Can have large impact when work is small.
- Avoids rescheduling if $N_{threads} = N_{cores}$
- Also reduces randomness without extra thread

```
int main(){
...

for (i=0; i<nthreads-1; i++)
  pthread_create(&thread[i],&attr,leapfrog,(void*)&arg[i]);

leapfrog((void*)&arg[nthreads-1]);

for (i=0; i<nthreads-1; i++)
  pthread_join(thread[i],NULL);
...
```