Concepts in Multicore Programming

Lecture 7

Stencil Computations*

Charles E. Leiserson

February 25, 2010

*Slides courtesy of Matteo Frigo

Heat diffusion

1D heat diffusion equation:

$u(t,x)$: temperature at time t at position x.

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \ .$$

Finite difference approximation:

$$\begin{aligned}
\frac{\partial u}{\partial x}(t,x) &\approx \frac{u(t,x+\Delta x/2) - u(t,x-\Delta x/2)}{\Delta x} \\
\frac{\partial^2 u}{\partial x^2}(t,x) &\approx \frac{(\partial u/\partial x)(t,x+\Delta x/2) - (\partial u/\partial x)(t,x-\Delta x/2)}{\Delta x} \\
&\approx \frac{u(t,x+\Delta x) - 2u(t,x) + u(t,x-\Delta x)}{(\Delta x)^2} \ .
\end{aligned}$$

3-point stencil

Finite differences for the heat diffusion equation:

$$\frac{u(t+1,x_i)-u(t,x_i)}{\Delta t} = \frac{u(t,x_{i-1})-2u(t,x_i)+u(t,x_{i+1})}{(\Delta x)^2} \ .$$

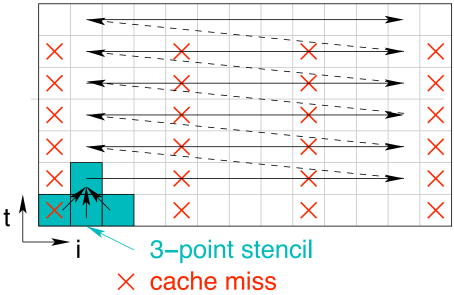
Simple implementation:

```
for (t = 0; t < T; ++t) {                    /* time loop */
    u[(t+1)%2][0] = left_boundary();
    for (i = 1; i < N - 1; ++i)              /* space loop */
        u[(t+1)%2][i] =
           kernel(u[t%2][i-1], u[t%2][i], u[t%2][i+1]);
    u[(t+1)%2][N - 1] = right_boundary();
}

double kernel(u_{i-1}, u_i, u_{i+1})
{
    return u_i + Δt/(Δx)^2 * (u_{i-1} - 2*u_i + u_{i+1});
}
```

3-point stencil on a cache

```
for (t = 0; t < T; ++t) {                /* time loop */
    u[(t+1)%2][0] = left_boundary();
    for (i = 1; i < N - 1; ++i)          /* space loop */
        u[(t+1)%2][i] =
           kernel(u[t%2][i-1], u[t%2][i], u[t%2][i+1]);
    u[(t+1)%2][N - 1] = right_boundary();
}
```



If array u is larger than the cache, the number of misses is proportional to the number of accesses.

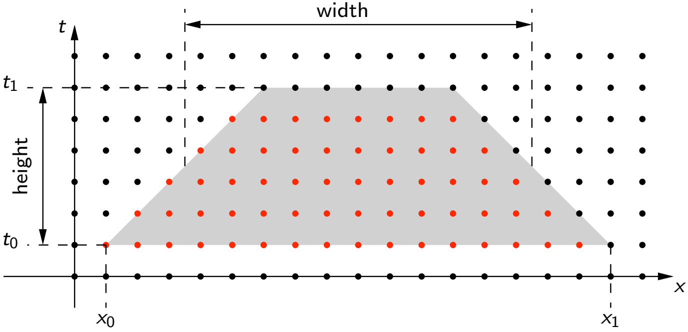
Cache oblivious algorithm for 3-point stencil

Recursively traverse trapezoidal regions of spacetime points (t, x) such that:

$$t_0 \leq t < t_1$$

$$x_0 + \dot{x}_0(t - t_0) \leq x < x_1 + \dot{x}_1(t - t_0)$$

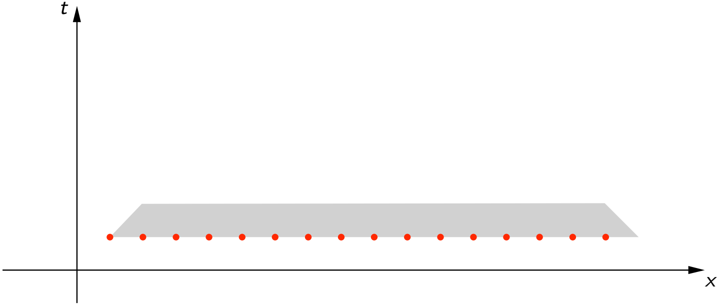
$$\dot{x}_i \in \{-1, 0, 1\}$$



Base case

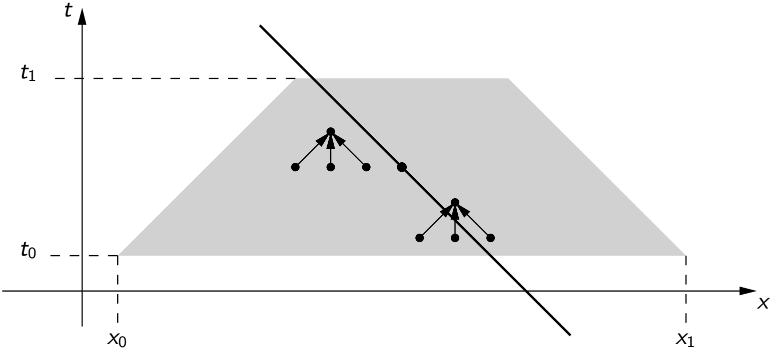
If height $= 1$, compute all spacetime points in the trapezoid.

Any order of computation is valid, because these points do not depend upon each other.



Space cut

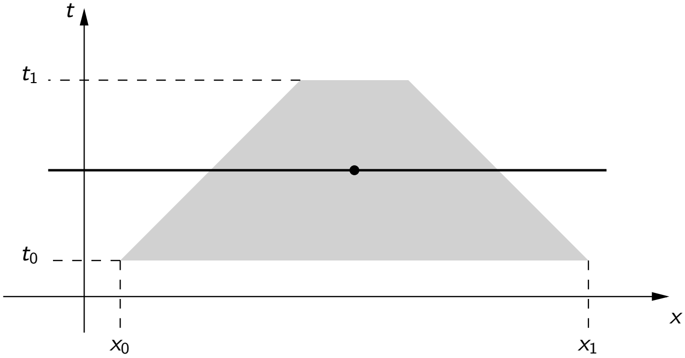
If width $\geq 2 \cdot$ height, cut the trapezoid with a line of slope -1 through the center.



Traverse first the trapezoid on the left, then the one on the right.

Time cut

If width $< 2 \cdot$ height, cut the trapezoid with a horizontal line through the center.



Traverse the bottom trapezoid first, then the top one.

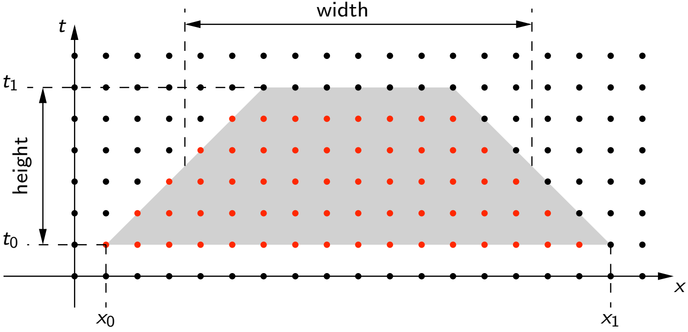
C implementation

```
void trapezoid(int t0, int t1, int x0, int ẋ0, int x1, int ẋ1)
{
  int Δt = t1 - t0;
  if (Δt == 1) {
    int x;
    for (x = x0; x < x1; ++x)
      kernel(t0, x);
  } else if (Δt > 1) {
    if (2 * (x1 - x0) + (ẋ1 - ẋ0) * Δt >= 4 * Δt) {
      int xm = (2 * (x0 + x1) + (2 + ẋ0 + ẋ1) * Δt) / 4;
      trapezoid(t0, t1, x0, ẋ0, xm, -1);
      trapezoid(t0, t1, xm, -1, x1, ẋ1);
    } else {
      int s = Δt / 2;
      trapezoid(t0, t0 + s, x0, ẋ0, x1, ẋ1);
      trapezoid(t0 + s, t1, x0 + ẋ0 * s, ẋ0, x1 + ẋ1 * s, ẋ1);
    }
  }
}
```

Cache complexity of the stencil algorithm

When width + height $= \Theta(Z)$:

- number of cache misses $= O(\text{width} + \text{height})$.
- number of points $= \Theta(\text{width} \cdot \text{height})$.
- Algorithm guarantees that height $= \Theta(\text{width})$.
- Thus, height $= \Theta(Z)$, width $= \Theta(Z)$.
- Thus, number of cache misses $= \Theta(\text{number of points}/Z)$.



Demo

Simulation:

- $\Delta x = 95$.
- $\Delta t = 87$.
- $\dot{x}_0 = \dot{x}_1 = 0$.
- LRU cache.
- Line size $=$ 4 points.
- Cache size $=$ 4, 8, 16, or 32 cache lines.
- Cache miss latency $=$ 10 cycles.