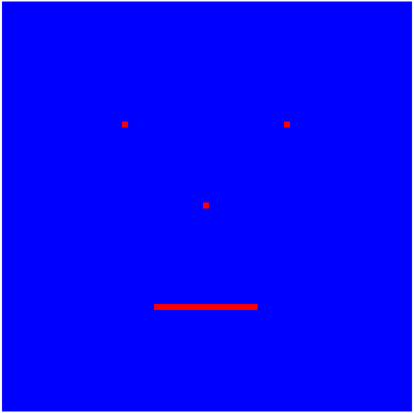
Parallel Computing I

Cluster: All-Reduce and All-To-All and Scan

Heat Distribution Problem

Suppose a thin metal plate has a temperature of $\mathbf{0}$ $^\circ\mathrm{C}$ along each edge with certain "hot spots" in the interior having a temperator of $\mathbf{100}$ $^\circ\mathrm{C}$.



What is the temperature everywhere else in the plate?

Heat Distribution Problem

Suppose a thin metal plate has a temperature of $\mathbf{0}\ {}^{\circ}\mathrm{C}$ along each edge with certain "hot spots" in the interior having a temperator of $\mathbf{100}\ {}^{\circ}\mathrm{C}$.

What is the temperature everywhere else in the plate?

Heat Distribution Problem: Laplace's Equation

Suppose a thin metal plate has a temperature of $0\ ^\circ\mathrm{C}$ along each edge with certain "hot spots" in the interior having a temperator of $100\ ^\circ\mathrm{C}$. What is the temperature everywhere else in the plate?

Let $h(x, y)$ be the temperature at point (x, y).

For edge points (x, y), $h(x, y) = 0$.
For hot-spot points (x, y), $h(x, y) = 100$.

For all other points, $h(x, y)$ satisfies *Laplace's equation*:

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = 0$$

Heat Distribution Problem: Laplace's Equation

Suppose a thin metal plate has a temperature of $0\ ^{\circ}\mathrm{C}$ along each edge with certain "hot spots" in the interior having a temperator of $100\ ^{\circ}\mathrm{C}$. What is the temperature everywhere else in the plate?

Let $h(x, y)$ be the temperature at point (x, y).

For edge points (x, y), $h(x, y) = 0$.
For hot-spot points (x, y), $h(x, y) = 100$.

For all other points, $h(x, y)$ satisfies *Laplace's equation*:

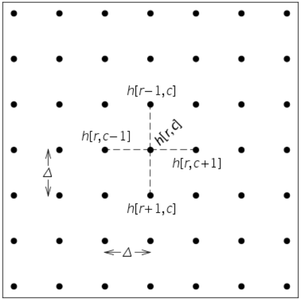
$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = 0$$

Solve this partial differential equation via a numerical integration algorithm.

Heat Distribution Problem: Mesh Points

Solve this partial differential equation via a numerical integration algorithm.
Approximate the continuous domain of the plate
by a discrete *mesh* of equally spaced points.

- H points in the y-direction
- W points in the x-direction
- $\boldsymbol{\Delta}$ distance between adjacent mesh points



Heat Distribution Problem: Laplace's Equation and Mesh Points

Solve this partial differential equation on the mesh points.

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = 0$$

Heat Distribution Problem: Laplace's Equation and Mesh Points

Solve this partial differential equation on the mesh points.

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = 0$$

$$\frac{\partial h}{\partial x} = \frac{\partial h_{\rightarrow}}{\partial x}$$

$$\frac{\partial h}{\partial x} = \frac{\partial h_{\leftarrow}}{\partial x}$$

$$\frac{\partial^2 h}{\partial x^2} = \frac{\partial}{\partial x}\frac{\partial h}{\partial x}$$

Heat Distribution Problem: Laplace's Equation and Mesh Points

Solve this partial differential equation on the mesh points.

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = 0$$

$$\frac{\partial h}{\partial x} = \frac{\partial h_{\rightarrow}}{\partial x} \approx \frac{h[r,c] - h[r,c-1]}{\Delta}$$

$$\frac{\partial h}{\partial x} = \frac{\partial h_{\leftarrow}}{\partial x} \approx \frac{h[r,c+1] - h[r,c]}{\Delta}$$

$$\frac{\partial^2 h}{\partial x^2} = \frac{\partial}{\partial x}\frac{\partial h}{\partial x}$$

Heat Distribution Problem: Laplace's Equation and Mesh Points

Solve this partial differential equation on the mesh points.

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = 0$$

$$\frac{\partial h}{\partial x} = \frac{\partial h_{\rightarrow}}{\partial x} \approx \frac{h[r,c]-h[r,c-1]}{\Delta}$$

$$\frac{\partial h}{\partial x} = \frac{\partial h_{\leftarrow}}{\partial x} \approx \frac{h[r,c+1]-h[r,c]}{\Delta}$$

$$\frac{\partial^2 h}{\partial x^2} = \frac{\partial}{\partial x}\frac{\partial h}{\partial x} \approx \frac{\frac{\partial h_{\leftarrow}}{\partial x} - \frac{\partial h_{\rightarrow}}{\partial x}}{\Delta} \approx \frac{\frac{h[r,c+1]-h[r,c]}{\Delta} - \frac{h[r,c]-h[r,c-1]}{\Delta}}{\Delta}$$

Heat Distribution Problem: Laplace's Equation and Mesh Points

Solve this partial differential equation on the mesh points.

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = 0$$

$$\frac{\partial h}{\partial x} = \frac{\partial h_{\rightarrow}}{\partial x} \approx \frac{h[r,c]-h[r,c-1]}{\Delta}$$

$$\frac{\partial h}{\partial x} = \frac{\partial h_{\leftarrow}}{\partial x} \approx \frac{h[r,c+1]-h[r,c]}{\Delta}$$

$$\frac{\partial^2 h}{\partial x^2} = \frac{\partial}{\partial x}\frac{\partial h}{\partial x} \approx \frac{\frac{\partial h_{\leftarrow}}{\partial x} - \frac{\partial h_{\rightarrow}}{\partial x}}{\Delta} \approx \frac{h[r,c+1]+h[r,c-1]-2h[r,c]}{\Delta^2}$$

Heat Distribution Problem: Laplace's Equation and Mesh Points

Solve this partial differential equation on the mesh points.

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = 0$$

$$\frac{\partial h}{\partial x} = \frac{\partial h_{\rightarrow}}{\partial x} \approx \frac{h[r,c] - h[r,c-1]}{\Delta}$$

$$\frac{\partial h}{\partial x} = \frac{\partial h_{\leftarrow}}{\partial x} \approx \frac{h[r,c+1] - h[r,c]}{\Delta}$$

$$\frac{\partial^2 h}{\partial x^2} = \frac{\partial}{\partial x}\frac{\partial h}{\partial x} \approx \frac{\frac{\partial h_{\leftarrow}}{\partial x} - \frac{\partial h_{\rightarrow}}{\partial x}}{\Delta} \approx \frac{h[r,c+1] + h[r,c-1] - 2h[r,c]}{\Delta^2}$$

$$\frac{\partial^2 h}{\partial y^2} = \frac{\partial}{\partial y}\frac{\partial h}{\partial y} \approx \frac{\frac{\partial h_{\leftarrow}}{\partial y} - \frac{\partial h_{\rightarrow}}{\partial y}}{\Delta} \approx \frac{h[r+1,c] + h[r-1,c] - 2h[r,c]}{\Delta^2}$$

Heat Distribution Problem: Laplace's Equation and Mesh Points

Solve this partial differential equation on the mesh points.

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = 0$$

$$\frac{h[r,c+1]+h[r,c-1]-2h[r,c]}{\Delta^2} + \frac{h[r+1,c]+h[r-1,c]-2h[r,c]}{\Delta^2} = 0$$

$$h[r,c] = \frac{h[r,c+1]+h[r,c-1]+h[r+1,c]+h[r-1,c]}{4}$$

Heat Distribution Problem: Laplace's Equation and Mesh Points

Solve this partial differential equation on the mesh points.

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = 0$$

$$\frac{h[r,c+1]+h[r,c-1]-2h[r,c]}{\Delta^2} + \frac{h[r+1,c]+h[r-1,c]-2h[r,c]}{\Delta^2} = 0$$

$$h[r,c] = \frac{h[r,c+1]+h[r,c-1]+h[r+1,c]+h[r-1,c]}{4}$$

The temperature at every mesh point (except the edges and hot spots) is the average of the temperatures of the four neighboring mesh points.

Heat Distribution Problem: Laplace's Equation and Mesh Points

Solve this partial differential equation on the mesh points.

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = 0$$

$$\frac{h[r,c+1]+h[r,c-1]-2h[r,c]}{\Delta^2} + \frac{h[r+1,c]+h[r-1,c]-2h[r,c]}{\Delta^2} = 0$$

$$h[r,c] = \frac{h[r,c+1]+h[r,c-1]+h[r+1,c]+h[r-1,c]}{4}$$

The temperature at every mesh point (except the edges and hot spots) is the average of the temperatures of the four neighboring mesh points.

Heat Distribution Problem: Initial Program

$h = \texttt{double}[H+2][W+2]$
$h_{\mathrm{new}} = \texttt{double}[H+2][W+2]$

for (r, c) **in** $(0..H{+}1, 0..W{+}1)$
    if $\texttt{isHotSpot}(r, c)$
        $h[r, c] = h_{\mathrm{new}}[r, c] = \texttt{tempHotSpot}(r, c)$
    else
        $h[r, c] = h_{\mathrm{new}}[r, c] = 0$
do
    for (r, c) **in** $(1..H, 1..W)$
        if $\neg\texttt{isHotSpot}(r, c)$
            $h_{\mathrm{new}}[r, c] = (h[r, c{+}1] + h[r, c{-}1] + h[r{+}1, c] + h[r{-}1, c])/4$
    $h = h_{\mathrm{new}}$
until ???

Heat Distribution Problem: Initial Program

$h = \texttt{double}[H+2][W+2]$
$h_{\text{new}} = \texttt{double}[H+2][W+2]$

for (r, c) **in** $(0..H{+}1, 0..W{+}1)$
    if $\texttt{isHotSpot}(r, c)$
        $h[r, c] = h_{\text{new}}[r, c] = \texttt{tempHotSpot}(r, c)$
    else
        $h[r, c] = h_{\text{new}}[r, c] = 0$
do
    for (r, c) **in** $(1..H, 1..W)$
        if $\neg\texttt{isHotSpot}(r, c)$
            $h_{\text{new}}[r, c] = (h[r, c{+}1] + h[r, c{-}1] + h[r{+}1, c] + h[r{-}1, c])/4$
    $h = h_{\text{new}}$
until ???

An example of a *relaxation algorithm*:
points "relax" from initial state and converge on solution.

Heat Distribution Problem: Initial Program

$$
\begin{array}{l}
h = \texttt{double}[H+2][W+2] \\
h_{\text{new}} = \texttt{double}[H+2][W+2] \\
\\
\textbf{for } (r, c) \textbf{ in } (0..H{+}1, 0..W{+}1) \\
\quad \textbf{if } \texttt{isHotSpot}(r, c) \\
\quad\quad h[r, c] = h_{\text{new}}[r, c] = \texttt{tempHotSpot}(r, c) \\
\quad \textbf{else} \\
\quad\quad h[r, c] = h_{\text{new}}[r, c] = 0 \\
\textbf{do} \\
\quad \textbf{for } (r, c) \textbf{ in } (1..H, 1..W) \\
\quad\quad \textbf{if } \neg\texttt{isHotSpot}(r, c) \\
\quad\quad\quad h_{\text{new}}[r, c] = (h[r, c{+}1] + h[r, c{-}1] + h[r{+}1, c] + h[r{-}1, c])/4 \\
\quad h = h_{\text{new}} \\
\textbf{until } ???
\end{array}
$$

An example of a *relaxation algorithm*:
points “relax” from initial state and converge on solution.
But, when does the algorithm terminate?

Heat Distribution Problem: Residuals

$$h_{\mathrm{new}}[r,c] = \frac{h[r,c+1]+h[r,c-1]+h[r+1,c]+h[r-1,c]}{4}$$

Heat Distribution Problem: Residuals

$$h_{\text{new}}[r,c] = \frac{h[r,c+1]+h[r,c-1]+h[r+1,c]+h[r-1,c]}{4}$$

$$h_{\text{new}}[r,c] = h[r,c] + \frac{h[r,c+1]+h[r,c-1]+h[r+1,c]+h[r-1,c]}{4} - h[r,c]$$

Heat Distribution Problem: Residuals

$$h_{\text{new}}[r,c] = \frac{h[r,c+1]+h[r,c-1]+h[r+1,c]+h[r-1,c]}{4}$$

$$h_{\text{new}}[r,c] = h[r,c] + \frac{h[r,c+1]+h[r,c-1]+h[r+1,c]+h[r-1,c]}{4} - h[r,c]$$

$$h_{\text{new}}[r,c] = h[r,c] + \frac{h[r,c+1]+h[r,c-1]+h[r+1,c]+h[r-1,c]-4h[r,c]}{4}$$

Heat Distribution Problem: Residuals

$$h_{\text{new}}[r,c] = \frac{h[r,c+1]+h[r,c-1]+h[r+1,c]+h[r-1,c]}{4}$$

$$h_{\text{new}}[r,c] = h[r,c] + \frac{h[r,c+1]+h[r,c-1]+h[r+1,c]+h[r-1,c]}{4} - h[r,c]$$

$$h_{\text{new}}[r,c] = h[r,c] + \frac{h[r,c+1]+h[r,c-1]+h[r+1,c]+h[r-1,c]-4h[r,c]}{4}$$

$$\xi[r,c] = h[r,c+1] + h[r,c-1] + h[r+1,c] + h[r-1,c] - 4h[r,c]$$
$$h_{\text{new}}[r,c] = h[r,c] + \frac{\xi[r,c]}{4}$$

$\xi[r,c]$ is the mesh point's *residual*:
the difference between its old and new values.

Heat Distribution Problem: Residuals

$$\xi[r,c] = h[r,c{+}1] + h[r,c{-}1] + h[r{+}1,c] + h[r{-}1,c] - 4h[r,c]$$
$$h_{\text{new}}[r,c] = h[r,c] + \frac{\xi[r,c]}{4}$$

$\xi[r,c]$ is the mesh point's *residual*:
the difference between its old and new values.

Heat Distribution Problem: Residuals

$$\xi[r,c] = h[r,c{+}1] + h[r,c{-}1] + h[r{+}1,c] + h[r{-}1,c] - 4h[r,c]$$
$$h_{\text{new}}[r,c] = h[r,c] + \tfrac{\xi[r,c]}{4}$$

$\xi[r,c]$ is the mesh point's *residual*:
the difference between its old and new values.

$$\Xi_{||} = \sum_{r=1}^{H} \sum_{c=1}^{W} \begin{cases} |\xi[r,c]| & \textbf{if } \neg \texttt{isHotSpot}(r,c) \\ \mathbf{0} & \textbf{if } \texttt{isHotSpot}(r,c) \end{cases}$$

$\Xi_{||}$ is the *total absolute residual*.

Heat Distribution Problem: Residuals

$$\xi[r,c] = h[r,c{+}1] + h[r,c{-}1] + h[r{+}1,c] + h[r{-}1,c] - 4h[r,c]$$
$$h_{\text{new}}[r,c] = h[r,c] + \tfrac{\xi[r,c]}{4}$$

$\xi[r,c]$ is the mesh point's *residual*:
the difference between its old and new values.

$$\Xi_{||} = \sum_{r=1}^{H}\sum_{c=1}^{W}\begin{cases} |\xi[r,c]| & \text{if } \neg\texttt{isHotSpot}(r,c) \\ 0 & \text{if } \texttt{isHotSpot}(r,c) \end{cases}$$

$\Xi_{||}$ is the *total absolute residual*.
At algorithm start, $\Xi_{||}$ is large. At solution, $\Xi_{||}$ is 0.
Thus, $\Xi_{||}$ is a measure of how the current state is from the solution state.

Heat Distribution Problem: Residuals

$$\xi[r,c] = h[r,c{+}1] + h[r,c{-}1] + h[r{+}1,c] + h[r{-}1,c] - 4h[r,c]$$
$$h_{\text{new}}[r,c] = h[r,c] + \tfrac{\xi[r,c]}{4}$$

$\xi[r,c]$ is the mesh point's *residual*:
the difference between its old and new values.

$$\Xi_{||} = \sum_{r=1}^{H} \sum_{c=1}^{W} \begin{cases} |\xi[r,c]| & \text{if } \neg\texttt{isHotSpot}(r,c) \\ 0 & \text{if } \texttt{isHotSpot}(r,c) \end{cases}$$

$\Xi_{||}$ is the *total absolute residual*.
At algorithm start, $\Xi_{||}$ is large. At solution, $\Xi_{||}$ is 0.
Thus, $\Xi_{||}$ is a measure of how the current state is from the solution state.
Terminate algorithm when $\Xi_{||}$ is small enough.

Heat Distribution Problem: Residuals

Terminate algorithm when $\Xi_{||}$ is small enough.

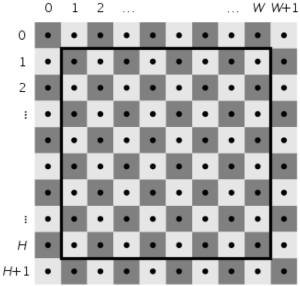
```
h = double[H + 2][W + 2]
h_new = double[H + 2][W + 2]
ε = 0.001

for (r, c) in (0..H+1, 0..W+1)
    if isHotSpot(r, c)
        h[r, c] = h_new[r, c] = tempHotSpot(r, c)
    else
        h[r, c] = h_new[r, c] = 0
Ξ_||^init = 0
for (r, c) in (1..H, 1..W)
    if ¬isHotSpot(r, c)
        ξ = h[r, c+1] + h[r, c−1] + h[r+1, c] + h[r−1, c] − 4h[r, c]
        Ξ_||^init = Ξ_||^init + |ξ|
do
    Ξ_|| = 0
    for (r, c) in (1..H, 1..W)
        if ¬isHotSpot(r, c)
            ξ = h[r, c+1] + h[r, c−1] + h[r+1, c] + h[r−1, c] − 4h[r, c]
            Ξ_|| = Ξ_|| + |ξ|
            h_new[r, c] = h[r, c] + ξ/4
    h = h_new
until Ξ_|| < εΞ_||^init
```

In practice, put an upper limit on interations
to ensure termination even if $\Xi_{||}$ is not converging.

Heat Distribution Problem: Red-Black Mesh Updating

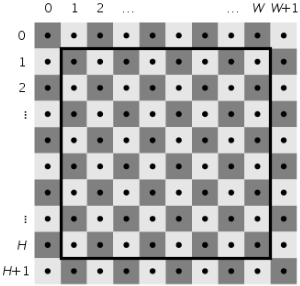
Suppose the mesh points are colored red and black, like a checkerboard.



New value of a red point only depends upon neighboring black points.
New value of a black point only depends upon neighboring red points.

Heat Distribution Problem: Red-Black Mesh Updating

Suppose the mesh points are colored red and black, like a checkerboard.

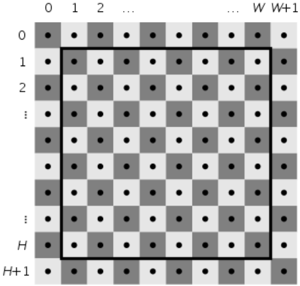


New value of a red point only depends upon neighboring black points.
New value of a black point only depends upon neighboring red points.

Alternate updating red points with updating black points.

Heat Distribution Problem: Red-Black Mesh Updating

Suppose the mesh points are colored red and black, like a checkerboard.



New value of a red point only depends upon neighboring black points.
New value of a black point only depends upon neighboring red points.

Alternate updating red points with updating black points.
Eliminate $\boldsymbol{h}_{\mathbf{new}}$.

Heat Distribution Problem: Red-Black Mesh Updating

Alternate updating red points with updating black points.

$\boldsymbol{h} = \texttt{double}[H + 2][W + 2]$
$\epsilon = 0.001$

for (r, c) **in** $(0..H{+}1, 0..W{+}1)$
    if $\texttt{isHotSpot}(r,c)$
        $h[r,c] = \texttt{tempHotSpot}(r,c)$
    else
        $h[r,c] = 0$
$\Xi_{||}^{\mathbf{init}} = 0$
for (r, c) **in** $(1..H, 1..W)$
    if $\neg\texttt{isHotSpot}(r,c)$
        $\xi = h[r, c{+}1] + h[r, c{-}1] + h[r{+}1, c] + h[r{-}1, c] - 4h[r, c]$
        $\Xi_{||}^{\mathbf{init}} = \Xi_{||}^{\mathbf{init}} + |\xi|$
do
    $\Xi_{||} = 0$
    for (r, c) **in** $(1..H, 1 + (r\&1)..W$ **by** $2)$
        if $\neg\texttt{isHotSpot}(r,c)$
            $\xi = h[r, c{+}1] + h[r, c{-}1] + h[r{+}1, c] + h[r{-}1, c] - 4h[r, c]$
            $\Xi_{||} = \Xi_{||} + |\xi|$
            $h[r, c] = h[r, c] + \xi/4$
    for (r, c) **in** $(1..H, 2 - (r\&1)..W$ **by** $2)$
        if $\neg\texttt{isHotSpot}(r,c)$
            $\xi = h[r, c{+}1] + h[r, c{-}1] + h[r{+}1, c] + h[r{-}1, c] - 4h[r, c]$
            $\Xi_{||} = \Xi_{||} + |\xi|$
            $h[r, c] = h[r, c] + \xi/4$
until $\Xi_{||} < \epsilon\Xi_{||}^{\mathbf{init}}$

Heat Distribution Problem: Successive Overrelaxation

Previous algorithm is $O(n^4)$:

- $O(n^2)$ per iteration
- $O(n^2)$ iterations to converge (due to the "small" correction $\xi[r, c]/4$)

Heat Distribution Problem: Successive Overrelaxation

Previous algorithm is $O(n^4)$:

- $O(n^2)$ per iteration
- $O(n^2)$ iterations to converge (due to the "small" correction $\xi[r, c]/4$)

Use a *relaxation parameter* ω to *overcorrect* at each iteration:

$$h[r, c] = h[r, c] + \omega \frac{\xi[r, c]}{4}$$

Heat Distribution Problem: Successive Overrelaxation

Previous algorithm is $O(n^4)$:

- $O(n^2)$ per iteration
- $O(n^2)$ iterations to converge (due to the "small" correction $\xi[r, c]/4$)

Use a *relaxation parameter* ω to *overcorrect* at each iteration:

$$h[r, c] = h[r, c] + \omega \frac{\xi[r, c]}{4}$$

Optimum ω is related to the *spectral radius* ρ_s:

$$\omega = \frac{2}{1 + \sqrt{1 - \rho_s^2}} \qquad \rho_s = \frac{\cos(\pi/H) + \cos(\pi/W)}{2}$$

Heat Distribution Problem: Successive Overrelaxation

Previous algorithm is $O(n^4)$:

- $O(n^2)$ per iteration
- $O(n^2)$ iterations to converge (due to the "small" correction $\xi[r, c]/4$)

Use a *relaxation parameter* ω to *overcorrect* at each iteration:

$$h[r, c] = h[r, c] + \omega \frac{\xi[r, c]}{4}$$

Optimum ω is related to the *spectral radius* ρ_s:

$$\omega = \frac{2}{1 + \sqrt{1 - \rho_s^2}} \qquad \rho_s = \frac{\cos(\pi/H) + \cos(\pi/W)}{2}$$

New algorithm is $O(n^3)$: $O(n^2)$ per iteration, $O(n)$ iterations to converge.

Heat Distribution Problem: Chebyshev Acceleration

Can further reduce number of iterations,
by allowing ω to approach optimal value.

Heat Distribution Problem: Chebyshev Acceleration

Can further reduce number of iterations,
by allowing ω to approach optimal value.

For the first iteration, red half-sweep:

$$\omega = 1$$

For the first iteration, black half-sweep:

$$\omega = \frac{1}{1 - \rho_s^2/2}$$

For other iterations, for each half-sweep:

$$\omega = \frac{1}{1 - \rho_s^2 \omega/4}$$

HotSpotSeq.java

`code/HotSpotSeq.java`

- *imagefile* — output PJG image file name
- *H* — number of mesh rows (not including boundaries)
- *C* — number of mesh cols (not including boundaries)
- zero or more of
  - *rl* — lower row index of a hot sopt
  - *cl* — lower col index of a hot sopt
  - *ru* — upper row index of a hot sopt
  - *cu* — upper col index of a hot sopt
  - *temp* — temperature of a hot sopt

Heat Distribution on a Cluster

How can we parallelize this algorithm for a cluster parallel computer?

Heat Distribution on a Cluster

How can we parallelize this algorithm for a cluster parallel computer?

Partition the H rows among the K processes;
each process responsible for H/K rows.

- Each process updates mesh points for its own rows
- Each process calculates $\Xi_{||}$ for its own rows

- Any concerns about load balance?

Heat Distribution on a Cluster

Partition the H rows among the K processes;
each process responsible for H/K rows.

Each process calculates $\Xi_{||}$ for its own rows.

But each process needs $\Xi_{||}$ for all rows for termination check.

- Initial $\Xi_{||}$ at program start.
- $\Xi_{||}$ after each red-blck mesh update.

Which collective communication operation?

Heat Distribution on a Cluster

Partition the $\boldsymbol{H}$ rows among the $\boldsymbol{K}$ processes;
each process responsible for $\boldsymbol{H}/\boldsymbol{K}$ rows.

Each process calculates $\boldsymbol{\Xi}_{||}$ for its own rows.

But each process needs $\boldsymbol{\Xi}_{||}$ for all rows for termination check.

- Initial $\boldsymbol{\Xi}_{||}$ at program start.
- $\boldsymbol{\Xi}_{||}$ after each red-blck mesh update.

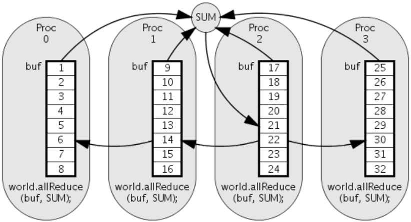
Which collective communication operation? all-reduce

All-Reduce

```
world.allReduce (buf, op);
```

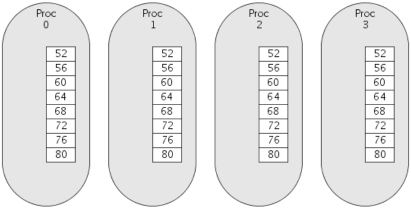
All-Reduce

```
world.allReduce (buf, op);
```



All-Reduce

```
world.allReduce (buf, op);
```



All-Reduce Implementation and Message Time Model

An all-reduce is equivalent to a reduction followed by a broadcast:

```
void allReduce (Buf buffer, Op op) {
  reduce (0, buffer, op);
  broadcast (0, buffer);
}
```

All-Reduce Implementation and Message Time Model

An all-reduce is equivalent to a reduction followed by a broadcast:

```
void allReduce (Buf buffer, Op op) {
  reduce (0, buffer, op);
  broadcast (0, buffer);
}
```

$$T_{\text{all-reduce}}(b, K) = 2(L + \tfrac{1}{B}b)\lceil \log_2 K \rceil$$

All-Reduce Implementation and Message Time Model

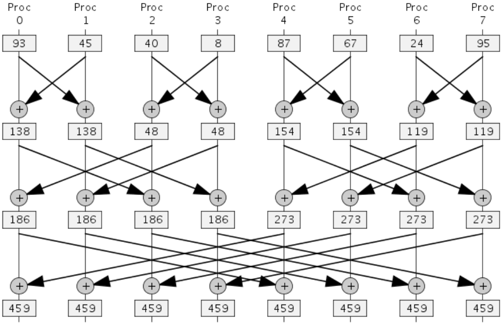
An all-reduce is equivalent to a reduction followed by a broadcast:

```
void allReduce (Buf buffer, Op op) {
  reduce (0, buffer, op);
  broadcast (0, buffer);
}
```

$$T_{\text{all-reduce}}(b, K) = 2(L + \tfrac{1}{B}b)\lceil \log_2 K \rceil$$

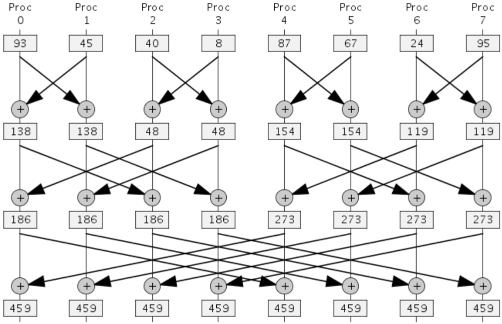
Can we do better?

All-Reduce Implementation and Message Time Model



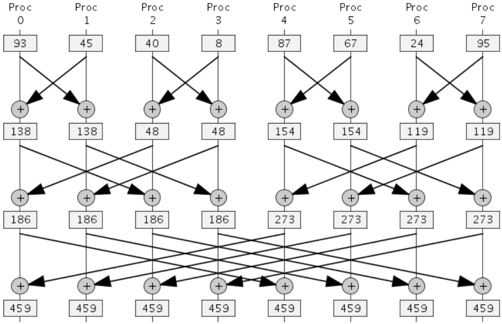
Each node simultaneously exchanges data buffer with another node and combines the received value with its own value.

All-Reduce Implementation and Message Time Model



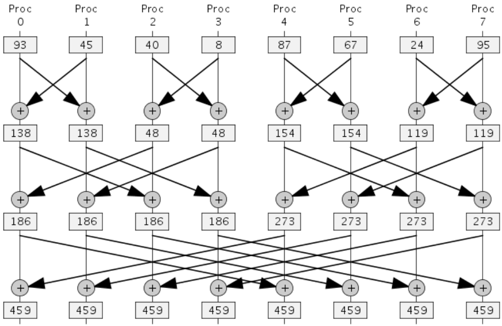
Each node simultaneously exchanges data buffer with another node and combines the received value with its own value.
First round, exchanges between nodes one rank apart.
Second round, exchanges between nodes two ranks apart.
Third round, exchanges between nodes four ranks apart.

All-Reduce Implementation and Message Time Model



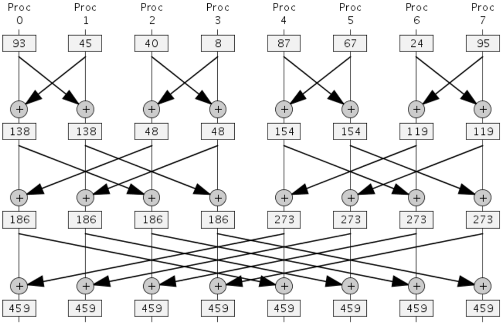
Each node simultaneously exchanges data buffer with another node and combines the received value with its own value.
N^{th} round, exchanges between nodes 2^{n-1} ranks apart.

All-Reduce Implementation and Message Time Model



$$T_{\textbf{all-reduce}}(b, K) =$$

All-Reduce Implementation and Message Time Model



$$T_{\text{all-reduce}}(b, K) = (L + \tfrac{1}{B}b)\lceil \log_2 K \rceil$$

Note: Assumes same latency and bandwith for both **1** send and K simultaneous sends.

Heat Distribution on a Cluster

Partition the H rows among the K processes;
each process responsible for H/K rows.
Each process updates mesh points for its own rows.

$$
\begin{array}{l}
\textbf{for } (r, c) \textbf{ in } (lbH..upB, 1 + (r\&1)..W \textbf{ by } 2) \\
\quad \textbf{if } \neg\texttt{isHotSpot}(r, c) \\
\quad\quad \xi = h[r, c{+}1] + h[r, c{-}1] + h[r{+}1, c] + h[r{-}1, c] - 4h[r, c] \\
\quad\quad \Xi_{||} = \Xi_{||} + |\xi| \\
\quad\quad h[r, c] = h[r, c] + \xi/4 \\
\textbf{for } (r, c) \textbf{ in } (lbH..upB, 2 - (r\&1)..W \textbf{ by } 2) \\
\quad \textbf{if } \neg\texttt{isHotSpot}(r, c) \\
\quad\quad \xi = h[r, c{+}1] + h[r, c{-}1] + h[r{+}1, c] + h[r{-}1, c] - 4h[r, c] \\
\quad\quad \Xi_{||} = \Xi_{||} + |\xi| \\
\quad\quad h[r, c] = h[r, c] + \xi/4
\end{array}
$$

What data is accessed by each process on each iteration?

Heat Distribution on a Cluster

Partition the H rows among the K processes;
each process responsible for H/K rows.
Each process updates mesh points for its own rows.

$$
\begin{array}{l}
\textbf{for } (r, c) \textbf{ in } (lbH..upB, 1 + (r\&1)..W \textbf{ by } 2) \\
\quad \textbf{if } \neg \texttt{isHotSpot}(r, c) \\
\qquad \xi = h[r, c{+}1] + h[r, c{-}1] + h[r{+}1, c] + h[r{-}1, c] - 4h[r, c] \\
\qquad \Xi_{||} = \Xi_{||} + |\xi| \\
\qquad h[r, c] = h[r, c] + \xi/4 \\
\textbf{for } (r, c) \textbf{ in } (lbH..upB, 2 - (r\&1)..W \textbf{ by } 2) \\
\quad \textbf{if } \neg \texttt{isHotSpot}(r, c) \\
\qquad \xi = h[r, c{+}1] + h[r, c{-}1] + h[r{+}1, c] + h[r{-}1, c] - 4h[r, c] \\
\qquad \Xi_{||} = \Xi_{||} + |\xi| \\
\qquad h[r, c] = h[r, c] + \xi/4
\end{array}
$$

What data is accessed by each process on each iteration?

Every process updates mesh points for its own rows.
But, every process reads one row of each neighboring process.

Heat Distribution on a Cluster

Partition the H rows among the K processes;
each process responsible for H/K rows.

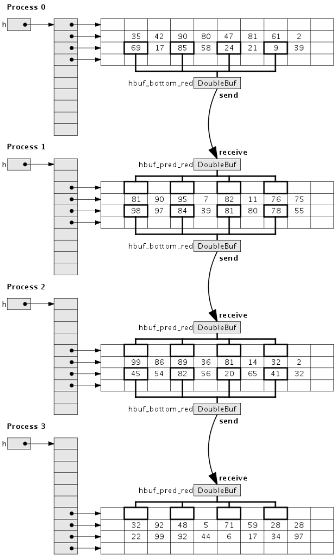
Each process updates mesh points for its own rows.
But, every process reads one row of each neighboring process.

Each process allocates its own rows,
plus two additional rows (one at top and one at bottom).

Additional rows will hold copies of mesh elements
from last row of previous process and from first row of next process.

Copy appropriate elements after each half-sweep.

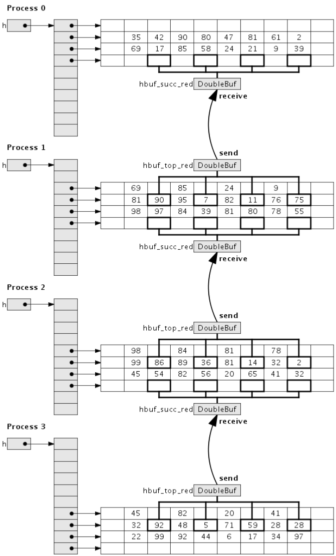
Mesh Element Allocation and Communication



After the red half-sweep:

- First, each process sends its last row of red elements forward, and receives the last row of the previous.
    - process with rank **0** sends
    - process with rank ***size*** − **1** receives
    - all other processes send-receive

Mesh Element Allocation and Communication

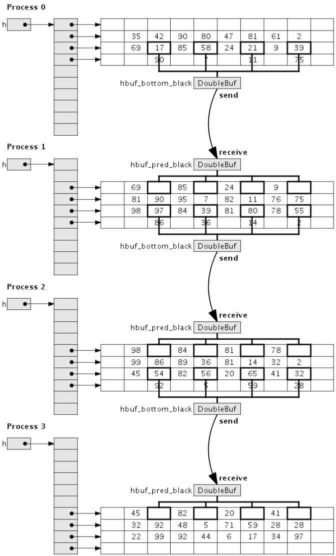


After the red half-sweep:

- First, each process sends its last row of red elements forward, and receives the last row of the previous.
  - process with rank **0** sends
  - process with rank ***size*** **−1** receives
  - all other processes send-receive
- Then, each process sends its first row of red elements backward, and receives the first row of the next.

```
DoubleBuf.sliceBuffer(h[r], new Range(1+(r&1),W,2))
```

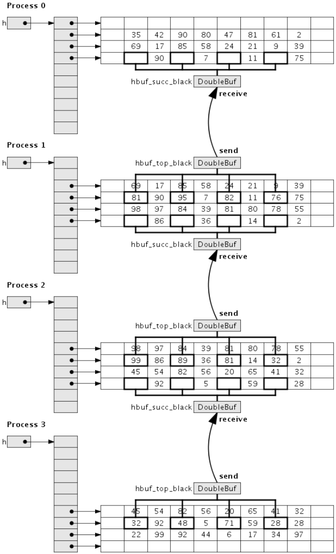
Mesh Element Allocation and Communication



After the black half-sweep:

- First, each process sends its last row of black elements forward, and receives the last row of the previous.

Mesh Element Allocation and Communication



After the black half-sweep:

- First, each process sends its last row of black elements forward, and receives the last row of the previous.
- Then, each process sends its first row of black elements backward, and receives the first row of the next.

```
DoubleBuf.sliceBuffer(h[r], new Range(2-(r&1),W,2))
```

HotSpotClu.java

code/HotSpotClu.java

`HotSpotClu.java`

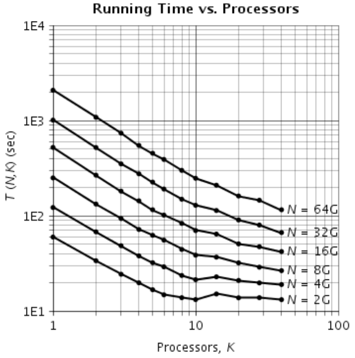
`code/HotSpotClu.java`

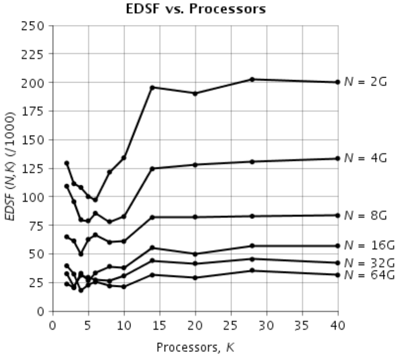
Homework 4: Derive a model to predict the running time.

What will you need?

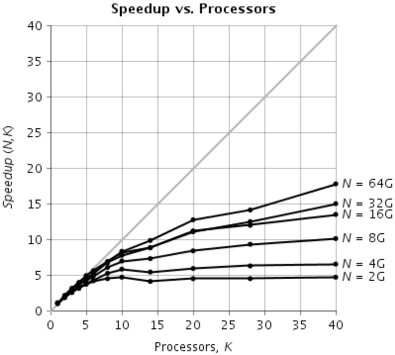
- a calculation time model
- a communication time model

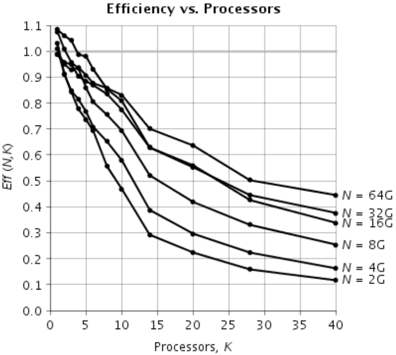
HotspotClu Running Time and EDSF





HotspotClu Speedup and Efficiency





Interestingly, classic Amdahl's Law behavior.

PRNGs and Statistical Tests

PRNGs do not generate truly-random numbers.
Often "good enough" for many applications (e.g. Monte Carlo algs.); nonetheless, helpful to have a measure of "how random" is a PRNG.

PRNGs and Statistical Tests

PRNGs do not generate truly-random numbers.
Often "good enough" for many applications (e.g. Monte Carlo algs.); nonetheless, helpful to have a measure of "how random" is a PRNG.

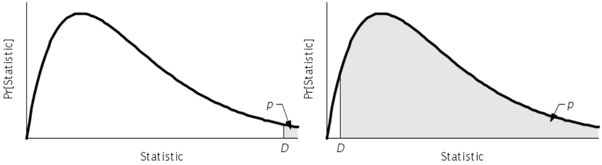
Statistical test of a PRNG:

1. Generate a large sample of random numbers
2. Calculate a *statistic* from the sampled numbers.
   Let D be the value of the statistic.
3. Calculate the statistic's p *value*.
   - the probability that the statistics' value would be $\geq D$, if the sampled numbers came from a truly random source.
4. If p is too large ($p > 0.999$) or too small ($p < 0.001$), then the PRNG fails the test.
   Otherwise, the PRNG passes the test.

PRNGs and Statistical Tests

3. Calculate the statistic's p *value*.
   - the probability that the statistics' value would be $\geq D$, if the sampled numbers came from a truly random source.
4. If p is too large ($p > 0.999$) or too small ($p < 0.001$), then the PRNG fails the test. Otherwise, the PRNG passes the test.

Example probability density function for a statistic;
p is the area under the pdf to the right of D.



Kolmogorov-Smirnov (K-S) Test

Define for the case of a random number source with a uniform distribution between 0 and 1.

Kolmogorov-Smirnov (K-S) Test

Define for the case of a random number source
with a uniform distribution between $\mathbf{0}$ and $\mathbf{1}$.

1. Generate a (large) sample of random numbers:

Kolmogorov-Smirnov (K-S) Test

Define for the case of a random number source with a uniform distribution between $\mathbf{0}$ and $\mathbf{1}$.

1. Generate a (large) sample of random numbers:

0.35612	**0.42731**	**0.90112**	**0.80018**	**0.47976**
0.81107	**0.61478**	**0.02314**	**0.69704**	**0.17270**

Kolmogorov-Smirnov (K-S) Test

2. Calculate the K-S statistic from the sampled numbers.

Kolmogorov-Smirnov (K-S) Test

2. Calculate the K-S statistic from the sampled numbers.
   a. Sort the sampled numbers in ascending order:

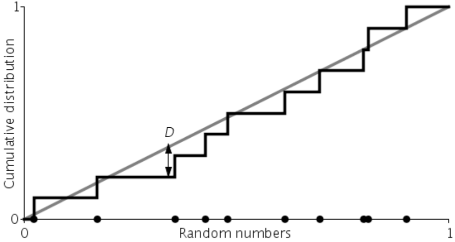
0.02314	**0.17270**	**0.35612**	**0.42731**	**0.47976**
0.61478	**0.69704**	**0.80018**	**0.81107**	**0.90112**

Kolmogorov-Smirnov (K-S) Test

2. Calculate the K-S statistic from the sampled numbers.
   a. Sort the sampled numbers in ascending order:

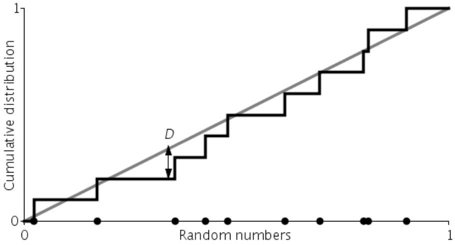
0.02314	**0.17270**	**0.35612**	**0.42731**	**0.47976**
0.61478	**0.69704**	**0.80018**	**0.81107**	**0.90112**

   b. Determine the cumulative distribution function for the sample. (Black curve in figure below.)
   Starts at $\mathbf{0}$ and jumps by $\mathbf{1/n}$ at each sampled number.



Kolmogorov-Smirnov (K-S) Test

2. Calculate the K-S statistic from the sampled numbers.
   c. Compare sample's cdf to the cdf for a uniform random variable. (Gray line in figure below.)
   d. K-S statistic $\boldsymbol{D}$ is the maximum absolute difference between the sample's cdf and the cdf for a uniform random variable.

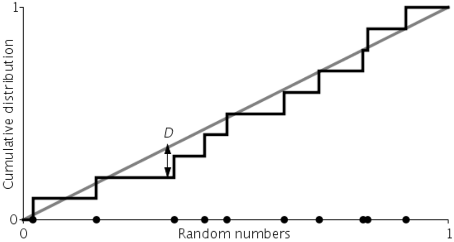


Kolmogorov-Smirnov (K-S) Test

2. Calculate the K-S statistic from the sampled numbers.
   d. K-S statistic $\boldsymbol{D}$ is the maximum absolute difference between the sample's cdf and the cdf for a uniform random variable.

$$D_i^- = \left|\frac{i}{n} - x_i\right| \qquad D_i^+ = \left|\frac{i+1}{n} - x_i\right|$$

$$D = \max \; \{D_i^-, D_i^+ \mid 0 \leq i \leq n-1\}$$



$$D = 0.15612$$

Kolmogorov-Smirnov (K-S) Test

3. Calculate the K-S statistic's p value.

$$p = P_{\mathrm{KS}}\left(\left[\sqrt{n} + 0.12 + \frac{0.11}{\sqrt{n}}\right] \cdot D\right)$$

$$P_{\mathrm{KS}}(u) = 2\sum_{i=1}^{\infty}(-1)^{i-1}e^{-2i^2u^2}$$

$$p = 0.95136$$

Kolmogorov-Smirnov (K-S) Test

3. Calculate the K-S statistic's p value.

$$p = P_{\mathrm{KS}}\left(\left[\sqrt{n} + 0.12 + \frac{0.11}{\sqrt{n}}\right] \cdot D\right)$$

$$P_{\mathrm{KS}}(u) = 2\sum_{i=1}^{\infty}(-1)^{i-1}e^{-2i^2u^2}$$

$$p = 0.95136$$

4. If p is too large ($p > 0.999$) or too small ($p < 0.001$), then the PRNG fails the test.
   Otherwise, the PRNG passes the test.

Block Ciphers as PRNGs

Recall the *counter-mode* PRNGs:

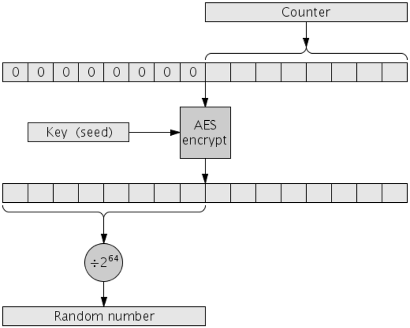
$$\begin{array}{l} \mathit{seed} \leftarrow \mathit{seed} + 1 \\ \textbf{return } H(\mathit{seed}) \end{array}$$

Block Ciphers as PRNGs

Recall the *counter-mode* PRNGs:

$$\begin{array}{l} \mathit{seed} \leftarrow \mathit{seed} + 1 \\ \mathrm{return}\ H(\mathit{seed}) \end{array}$$

Use a block cipher (like AES) as the hash function:



Block Ciphers as PRNGs

Use a block cipher (like AES) as the hash function.

Slower than a typical PRNG, but much more *secure*.

- Difficult to determine the PRNG's internal state.

Block Ciphers as PRNGs

Use a block cipher (like AES) as the hash function.

Slower than a typical PRNG, but much more *secure*.

- Difficult to determine the PRNG's internal state.

Is it sufficiently *random*?
Use the K-S statistical test.

Because test requires millions or billions of random numbers
and because AES encryption is a slow hash function,
want a program to perform K-S test in parallel.

AesTestSeq.java

code/AesTestSeq.java

- *key* — encryption key (**256**-bit)
- *n* — number of random numbers to sample

```
$ java AesTestSeq $KEY 60000000
N = 60000000
D = 9.070410274467089E-5
P = 0.7068995841396919
```

Parallel K-S Test Design

Three major time-consuming sections:

- Generate n random numbers.
- Sort the n random numbers.
- Iterate over the sorted random numbers to calculate D.

Ideally, parallelize each section.

Parallel K-S Test Design

Generate n random numbers.

Parallel K-S Test Design

Generate n random numbers. Easy!

Parallel K-S Test Design

Generate n random numbers. Easy!

Each process generates n/K random numbers.

Parallel K-S Test Design

Generate n random numbers. Easy!

Each process generates n/K random numbers.
Ensure that each process generates a *distinct* sequence of random numbers.

Parallel K-S Test Design

Generate n random numbers. Easy!

Each process generates n/K random numbers.
Ensure that each process generates a *distinct* sequence of random numbers.

Each process initializes its own counter
to the lower bound of its portion of the range 0 to $n-1$.

Process 0	.813	.723	.452	.263	.910	.438	.428	.204	.463	.685
Process 1	.028	.158	.588	.736	.698	.815	.975	.402	.234	.078
Process 2	.492	.284	.406	.695	.553	.424	.047	.224	.877	.582
Process 3	.346	.202	.439	.056	.095	.708	.497	.190	.572	.023

Parallel K-S Test Design

Generate n random numbers. Easy!

Each process generates n/K random numbers.
Ensure that each process generates a *distinct* sequence of random numbers.

Each process initializes its own counter
to the lower bound of its portion of the range $\mathbf{0}$ to $n - \mathbf{1}$.

Process 0	.813	.723	.452	.263	.910	.438	.428	.204	.463	.685
Process 1	.028	.158	.588	.736	.698	.815	.975	.402	.234	.078
Process 2	.492	.284	.406	.695	.553	.424	.047	.224	.877	.582
Process 3	.346	.202	.439	.056	.095	.708	.497	.190	.572	.023

Note: memory scalability

Parallel K-S Test Design

Sort the n random numbers.

Parallel K-S Test Design

Sort the n random numbers.

Each process sorts its own n/K random numbers.

Process 0	.204	.263	.428	.438	.452	.463	.685	.723	.813	.910
Process 1	.028	.078	.158	.234	.402	.588	.698	.736	.815	.975
Process 2	.047	.224	.284	.406	.424	.492	.553	.582	.695	.877
Process 3	.023	.056	.095	.190	.202	.346	.439	.497	.572	.708

Parallel K-S Test Design

Sort the n random numbers.

Each process sorts its own n/K random numbers.

Process 0	.204	.263	.428	.438	.452	.463	.685	.723	.813	.910
Process 1	.028	.078	.158	.234	.402	.588	.698	.736	.815	.975
Process 2	.047	.224	.284	.406	.424	.492	.553	.582	.695	.877
Process 3	.023	.056	.095	.190	.202	.346	.439	.497	.572	.708

But, this isn't the sort of *all* n random numbers.

After sorting, would like process $\mathbf{0}$ to have the n/K smallest numbers, process $\mathbf{1}$ to have the next n/K smallest numbers, ..., process $K - \mathbf{1}$ to have the n/K largest numbers.

Parallel K-S Test Design

Sort the n random numbers.

After sorting, would like process $\mathbf{0}$ to have the n/K smallest numbers,
process $\mathbf{1}$ to have the next n/K smallest numbers, $\ldots$,
process $K - \mathbf{1}$ to have the n/K largest numbers.

Somewhat difficult to evenly partition the sorted numbers.

Parallel K-S Test Design

Sort the n random numbers.

After sorting, would like process $\mathbf{0}$ to have the n/K smallest numbers, process $\mathbf{1}$ to have the next n/K smallest numbers, ..., process $K - \mathbf{1}$ to have the n/K largest numbers.

Somewhat difficult to evenly partition the sorted numbers.

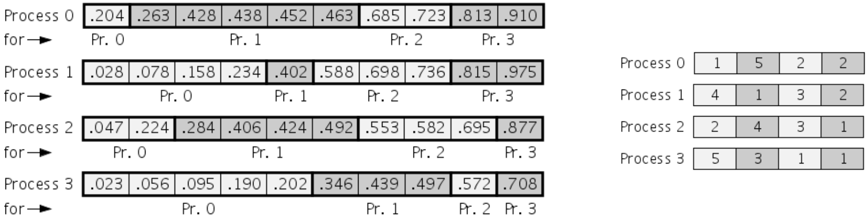
Assuming near uniform distribution, we can acheive something close:

After sorting, would like process $\mathbf{0}$ to have the numbers in $[0, 1/K)$, process $\mathbf{1}$ to have the numbers in $[1/K, 2/K)$, ..., process $K - \mathbf{1}$ to have the numbers in $[(K - 1)/K, 1)$.

Parallel K-S Test Design

Sort (and distribute) the ***n*** random numbers.

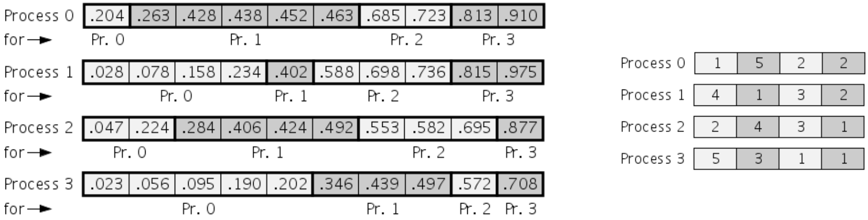
After sorting its own slice, each process can identify the sub-slices (and the lengths of the sub-slices) that belong to the other processes.



Parallel K-S Test Design

Sort (and distribute) the ***n*** random numbers.

After sorting its own slice, each process can identify the sub-slices (and the lengths of the sub-slices) that belong to the other processes.



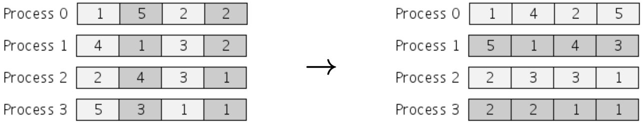
Now, each process knows length of data it will *send* to every other process, but each process does not know length of data it will *receive*.

Parallel K-S Test Design

Sort (and distribute) the n random numbers.

Now, each process knows length of data it will *send* to every other process, but each process does not know length of data it will *receive*.

Perform an all-to-all on the length information:

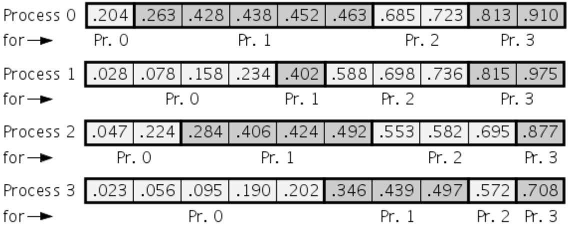


Now, each process knows the length of data it will *send* and *receive*.

Parallel K-S Test Design

Sort (and distribute) the n random numbers.

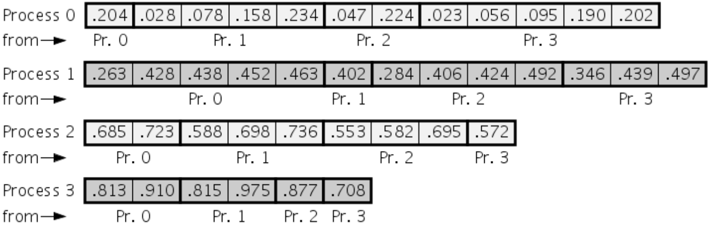
Now, each process knows the length of data it will *send* and *receive*.
Perform an all-to-all on the sorted random numbers:



Parallel K-S Test Design

Sort (and distribute) the ***n*** random numbers.

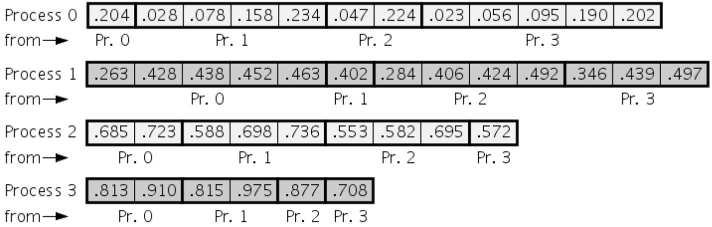
Now, each process knows the length of data it will *send* and *receive*. Perform an all-to-all on the sorted random numbers:



Parallel K-S Test Design

Sort (and distribute) the $\boldsymbol{n}$ random numbers.

Now, each process knows the length of data it will *send* and *receive*.
Perform an all-to-all on the sorted random numbers:



Each process sorts all of its received data.

Parallel K-S Test Design

Sort (and distribute) the ***n*** random numbers.

Now, each process knows the length of data it will *send* and *receive*.
Perform an all-to-all on the sorted random numbers:

Process 0	.023	.028	.047	.056	.078	.095	.158	.190	.202	.204	.224	.234	
Process 1	.263	.284	.346	.402	.406	.424	.428	.438	.439	.452	.463	.492	.497
Process 2	.553	.572	.582	.588	.685	.695	.698	.723	.736				
Process 3	.708	.813	.815	.877	.910	.975							

Parallel K-S Test Design

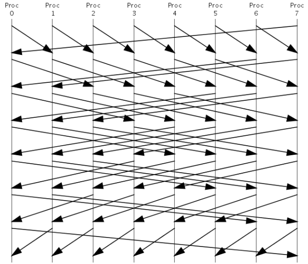
Sort (and distribute) the $\boldsymbol{n}$ random numbers.

Now, each process knows the length of data it will *send* and *receive*.
Perform an all-to-all on the sorted random numbers:

Process 0	.023	.028	.047	.056	.078	.095	.158	.190	.202	.204	.224	.234	
Process 1	.263	.284	.346	.402	.406	.424	.428	.438	.439	.452	.463	.492	.497
Process 2	.553	.572	.582	.588	.685	.695	.698	.723	.736				
Process 3	.708	.813	.815	.877	.910	.975							

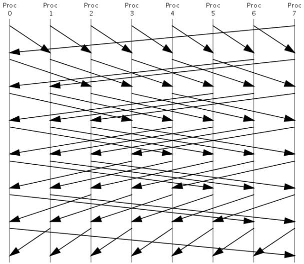
Processes may not have the same quantity of sorted data,
but, assuming uniform distribution, should be nearly balanced.

All-to-All Implementation and Message Time Model



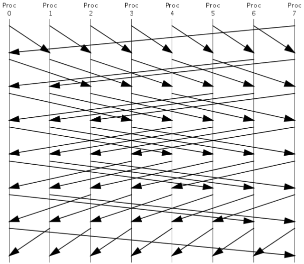
Each node simultaneously sends to one and receives from another.

All-to-All Implementation and Message Time Model



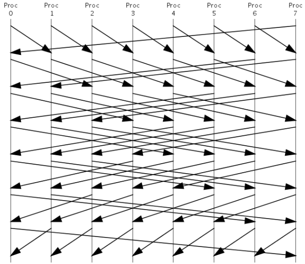
Each node simultaneously sends to one and receives from another.
First round, sends to one rank ahead and receives from one rank behind.
Second round, sends to two ranks ahead and receives from two ranks behind.
Third round, sends to three ranks ahead and receives from three ranks behind.

All-to-All Implementation and Message Time Model



Each node simultaneously sends to one and receives from another.
N^{th} round, sends to $n+1$ ranks ahead and receives from $n-1$ ranks behind.

All-to-All Implementation and Message Time Model



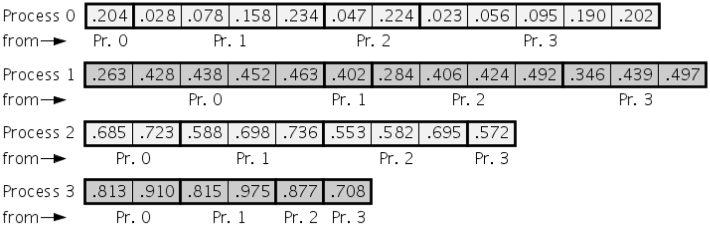
Each node simultaneously sends to one and receives from another.
N^{th} round, sends to $n+1$ ranks ahead and receives from $n-1$ ranks behind.

$$T_{\text{all-to-all}}(b, K) = (L + \frac{1}{B}b)(K-1)$$

Note: Assumes same latency and bandwith for both 1 send and K simultaneous sends.

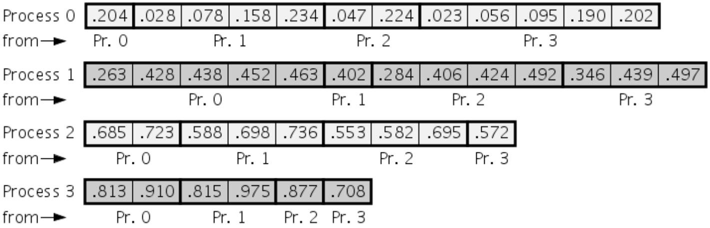
Parallel K-S Test Design

Iterate over the sorted random numbers to calculate ***D***.



Parallel K-S Test Design

Iterate over the sorted random numbers to calculate D.



Process **0** compares its first value to **0** and **1/40**.

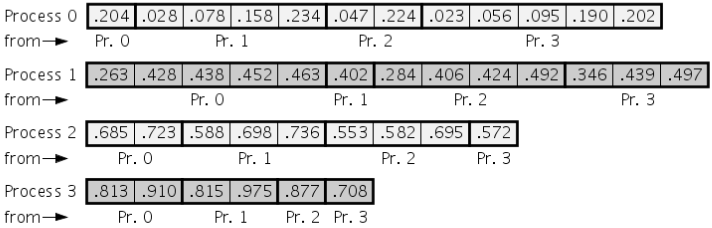
Process **0** compares its second value to **1/40** and **2/40**.

. . .

Process **0** compares its last value to **11/40** and **12/40**.

Parallel K-S Test Design

Iterate over the sorted random numbers to calculate D.



Process $\mathbf{1}$ compares its first value to $\mathbf{12/40}$ and $\mathbf{13/40}$.

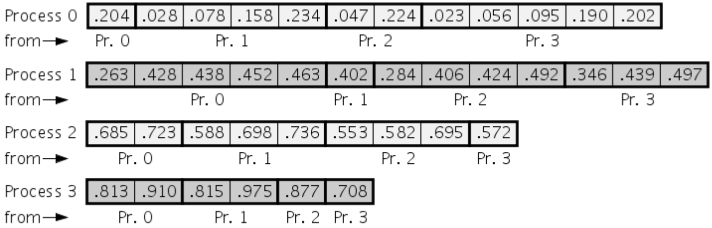
Process $\mathbf{1}$ compares its second value to $\mathbf{12/40}$ and $\mathbf{13/40}$.

...

Process $\mathbf{1}$ compares its last value to $\mathbf{24/40}$ and $\mathbf{25/40}$.

Parallel K-S Test Design

Iterate over the sorted random numbers to calculate D.



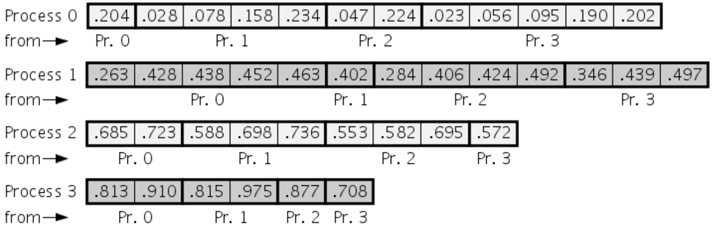
Process **2** compares its first value to $\mathbf{25/40}$ and $\mathbf{26/40}$.
Process **2** compares its second value to $\mathbf{26/40}$ and $\mathbf{27/40}$.

...

Process **2** compares its last value to $\mathbf{33/40}$ and $\mathbf{34/40}$.

Parallel K-S Test Design

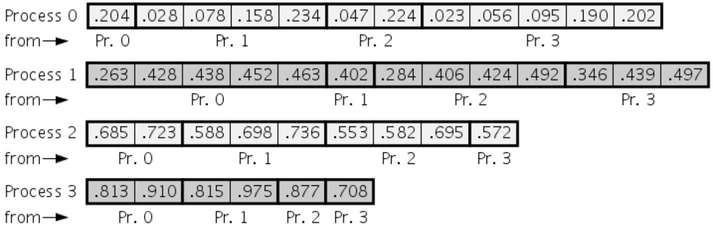
Iterate over the sorted random numbers to calculate ***D***.



Process **3** compares its first value to **34**/**40** and **35**/**40**.
Process **3** compares its second value to **35**/**40** and **36**/**40**.
. . .
Process **3** compares its last value to **39**/**40** and **40**/**40**.

Parallel K-S Test Design

Iterate over the sorted random numbers to calculate $\boldsymbol{D}$.



Process **3** compares its first value to **34**/**40** and **35**/**40**.
Process **3** compares its second value to **35**/**40** and **36**/**40**.
. . .
Process **3** compares its last value to **39**/**40** and **40**/**40**.

How does each process determine where its slice of the uniform cdf starts?

Parallel K-S Test Design

Iterate over the sorted random numbers to calculate $\boldsymbol{D}$.

How does each process determine where its slice of the uniform cdf starts?

Parallel K-S Test Design

Iterate over the sorted random numbers to calculate D.

How does each process determine where its slice of the uniform cdf starts?

Compare the i^{th} sorted random number with $\frac{i}{n}$ and $\frac{i+1}{n}$.

Each process needs to know how many sorted random numbers are less than its first sorted random number.

This is the *sum* of the *sizes* of sorted random numbers among the *previous* processes.

Which collective communication operation?

Parallel K-S Test Design

Iterate over the sorted random numbers to calculate D.

How does each process determine where its slice of the uniform cdf starts?

Compare the i^{th} sorted random number with $\frac{i}{n}$ and $\frac{i+1}{n}$.

Each process needs to know how many sorted random numbers
are less than its first sorted random number.

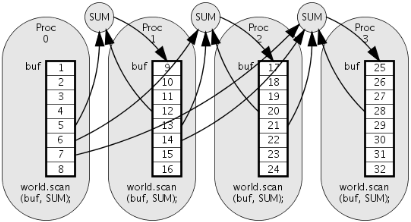
This is the *sum* of the *sizes* of sorted random numbers
among the *previous* processes.

Which collective communication operation? exclusive scan

- Why exclusive scan and not inclusive scan?

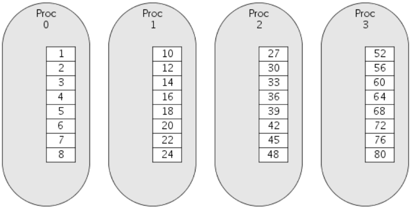
(Inclusive) Scan Implementation and Message Time Model

```
world.scan (buf, op);
```

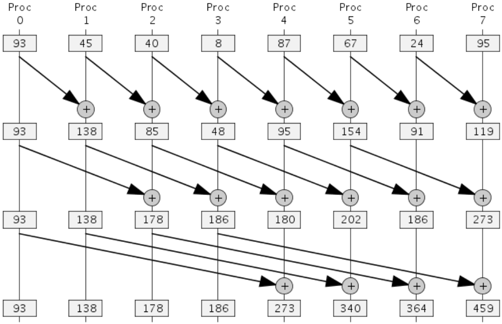


(Inclusive) Scan Implementation and Message Time Model

```
world.scan (buf, op);
```

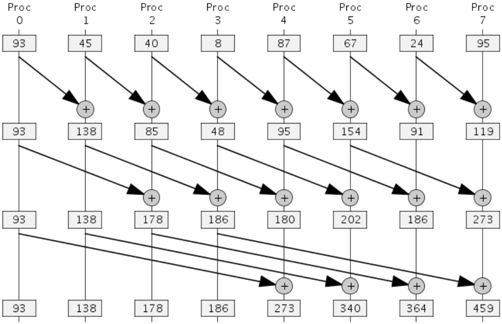


(Inclusive) Scan Implementation and Message Time Model



Each node only sends to higher-ranked ones and receives from lower-ranked ones; received messages are reduced with the node's current value.

(Inclusive) Scan Implementation and Message Time Model



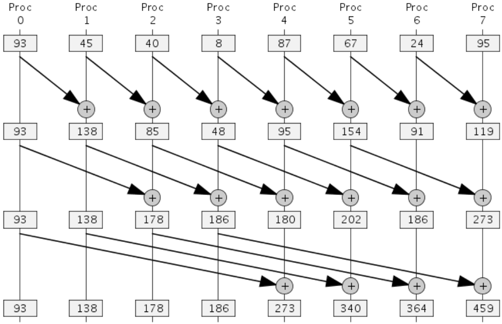
Each node only sends to higher-ranked ones and receives from lower-ranked ones; received messages are reduced with the node's current value.

First round, sends to one rank ahead.

Second round, sends to two ranks ahead.

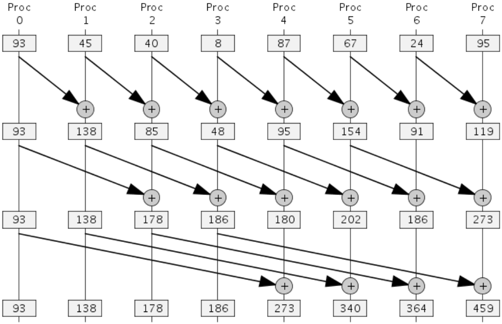
Third round, sends to four ranks ahead.

(Inclusive) Scan Implementation and Message Time Model



Each node only sends to higher-ranked ones and receives from lower-ranked ones; received messages are reduced with the node's current value.
N^{th} round, sends to 2^{n-1} ranks ahead.

(Inclusive) Scan Implementation and Message Time Model



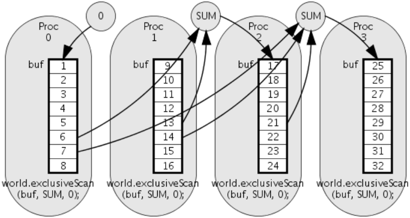
Each node only sends to higher-ranked ones and receives from lower-ranked ones; received messages are reduced with the node's current value.
$\boldsymbol{N}^{\text{th}}$ round, sends to $\mathbf{2}^{n-1}$ ranks ahead.

$$T_{\text{in-scan}}(b, K) = (L + \tfrac{1}{B} b) \lceil \log_2 K \rceil$$

Note: Assumes same latency and bandwith for both $\mathbf{1}$ send and K simultaneous sends.

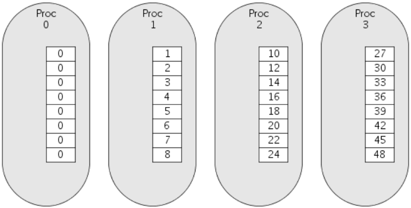
(Exclusive) Scan Implementation and Message Time Model

```
world.exclusiveScan (buf, op);
```

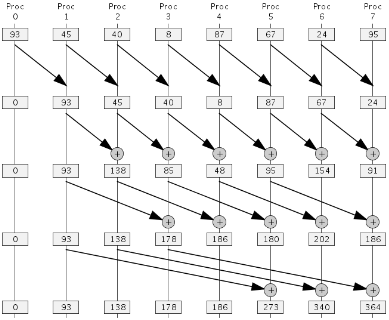


(Exclusive) Scan Implementation and Message Time Model

```
world.exclusiveScan (buf, op);
```

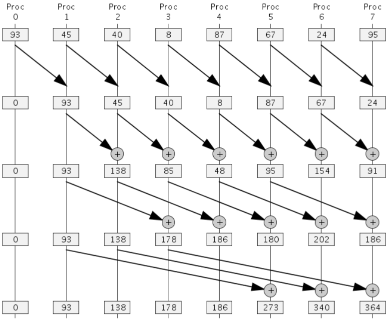


(Exclusive) Scan Implementation and Message Time Model



Each node first sends to one rank ahead;
received message replaces the node's current value.
Then perform an inclusive scan, but excluding rank **0**.

(Exclusive) Scan Implementation and Message Time Model



Each node first sends to one rank ahead;
received message replaces the node's current value.
Then perform an inclusive scan, but excluding rank **0**.

$$T_{\text{ex}-\text{scan}}(b, K) = (L + \tfrac{1}{B}b)(1 + \lceil \log_2 K \rceil)$$

Note: Assumes same latency and bandwith for both **1** send and K simultaneous sends.

Parallel K-S Test Design

Now, each process knows where its slice of the uniform cdf starts.

Iterate over the sorted random numbers to calculate D.

Parallel K-S Test Design

Now, each process knows where its slice of the uniform cdf starts.

Iterate over the sorted random numbers to calculate D.

Each process calculates the maximum of the D_i^- and D_i^+ for its own slice.

Process **0**	**0.066**
Process **1**	**0.128**
Process **2**	**0.137**
Process **3**	**0.167**

(Note: coincidence that per-process maximums are ascending by process.)

Parallel K-S Test Design

Now, each process knows where its slice of the uniform cdf starts.

Iterate over the sorted random numbers to calculate D.

Each process calculates the maximum of the D_i^- and D_i^+ for its own slice.

Process 0	0.066
Process 1	0.128
Process 2	0.137
Process 3	0.167

(Note: coincidence that per-process maximums are ascending by process.)

Reduce per-process maximum into process 0,
using the maximum reduction operator.

Process 0 computes and displays p value.

AesTestClu.java

code/AesTestClu.java

`AesTestClu.java`

`code/AesTestClu.java`

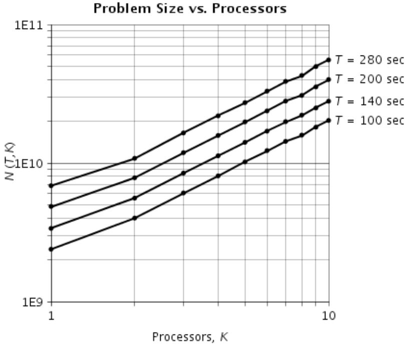
Scalability limited by memory to hold sample of random numbers.
JVM limit on size of an array means $n \approx 268M$ max for sequential,
but interested in problem sizes larger than that.

Collect data for sizeup rather than speedup.

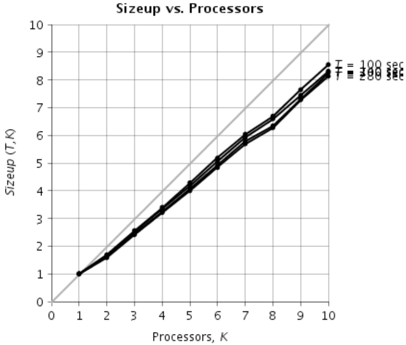
Running time dominated by $O(n \log n)$-time sorting steps.
Take problem size N to be $n \log n$.
Choose problem sizes assuming ideal sizeup.

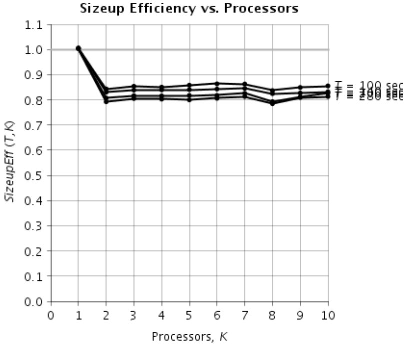
With these input sizes,
a $\mathtt{dr}N$ backend node has only sufficent memory to run one process;
can only scale up to $K = 10$.

AesTestClu Problem Size

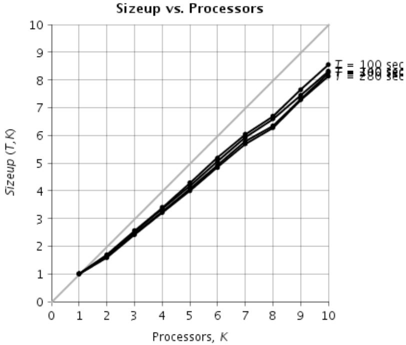


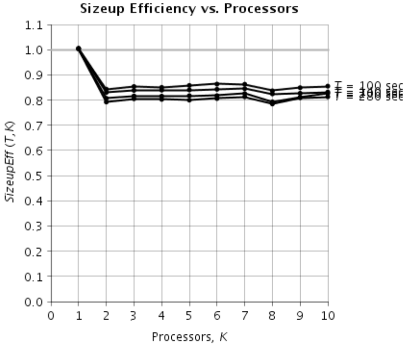
`AesTestClu` Sizeup and Sizeup Efficiency





AesTestClu Sizeup and Sizeup Efficiency





p values ranged from **0.11367** to **0.96589**.