Parallel Computing I

Cluster: All-Gather, Scalability and Pipelining, and Overlapping

N-body Problems

Predict the (past and/or future) motion of N objects,
given their motion at the present time.

- "motion" includes position as a function of time
- motion of one object influenced by external fields and other objects

N-body Problems

Predict the (past and/or future) motion of N objects,
given their motion at the present time.

- "motion" includes position as a function of time
- motion of one object influenced by external fields and other objects

Gravitational N-body problems:

- motion of astronomical bodies (planets, stars, star clusters, galaxies),
  due to mutual gravitational attraction

Electro-magnetic N-body problems:

- motion of charged particles (protons, antiprotons),
  due to mutual attraction/repulsion and magnetic fields

N-body Problems

Predict the (past and/or future) motion of N objects,
given their motion at the present time.

- "motion" includes position as a function of time
- motion of one object influenced by external fields and other objects

For systems of three or more objects, no analytic formula exists.
Instead, must simulate the motion.

N-body Problems

Predict the (past and/or future) motion of N objects,
given their motion at the present time.

- "motion" includes position as a function of time
- motion of one object influenced by external fields and other objects

For systems of three or more objects, no analytic formula exists.
Instead, must simulate the motion.

Simulations make simplifying assumptions:

- objects are idealized point particles (have mass and charge, but no volume)
- geometry assumed to be Euclidean
- motion and forces assumed to be Newtonian

N-body Problems

Predict the (past and/or future) motion of N objects,
given their motion at the present time.

- "motion" includes position as a function of time
- motion of one object influenced by external fields and other objects

For systems of three or more objects, no analytic formula exists.
Instead, must simulate the motion.

Simulations make simplifying assumptions:

- objects are idealized point particles (have mass and charge, but no volume)
- geometry assumed to be Euclidean
- motion and forces assumed to be Newtonian

Brief mathematics and physics review.

N-body Problems: Euclidean Geometry

Vectors:

- two-dimensional (2D) vectors, consist of (orthogonal) x and y components

$$\vec{a} = \langle a_x, a_y \rangle$$

Vector arithmetic:

- vector addition

$$\vec{a} + \vec{b} = \langle a_x + b_x, a_y + b_y \rangle$$

- vector subtraction

$$\vec{a} - \vec{b} = \langle a_x - b_x, a_y - b_y \rangle$$

- scalar product

$$c \cdot \vec{a} = \langle c \cdot a_x, c \cdot a_y \rangle$$

- vector magnitude

$$|\vec{a}| = \sqrt{a_x^2 + a_y^2}$$

N-body Problems: Newtonian Physics

Physical characteristics:

- mass of particle i

$$m_i$$

- position of particle i at time t

$$\vec{p}_i(t) = \langle p_{i,x}(t), p_{i,y}(t) \rangle$$

- velocity of particle i at time t

$$\vec{v}_i(t) = \langle v_{i,x}(t), v_{i,y}(t) \rangle$$

- acceleration of particle i at time t

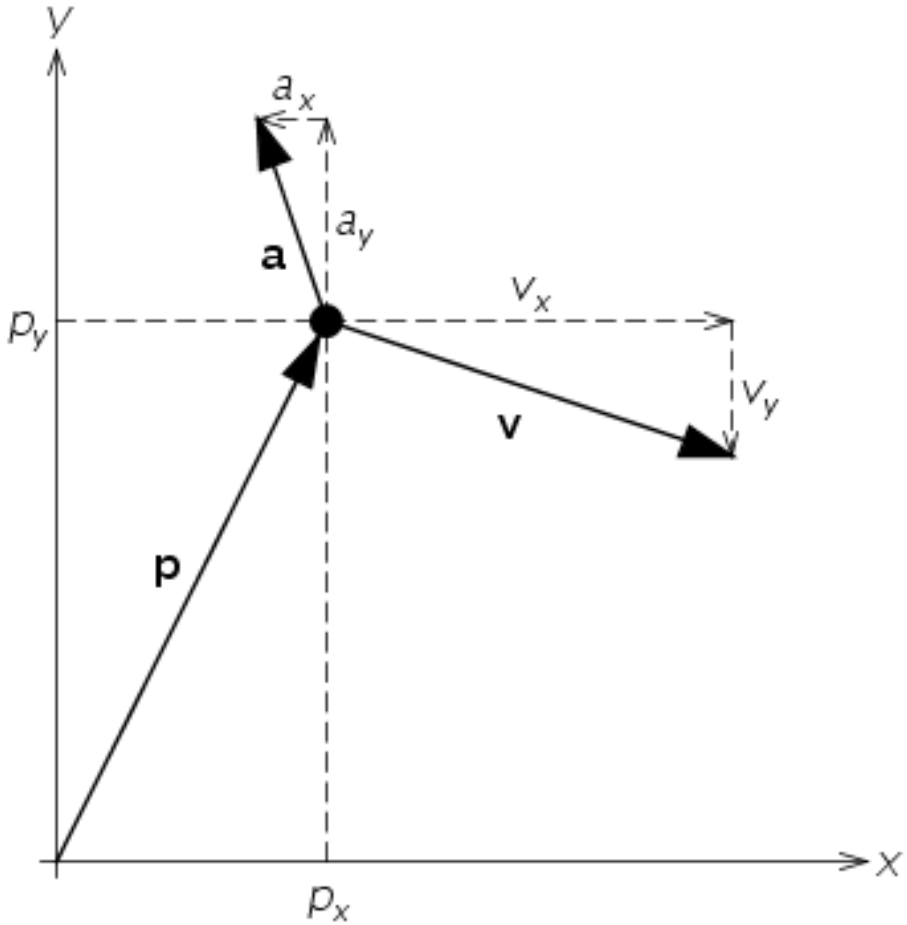
$$\vec{a}_i(t) = \langle a_{i,x}(t), a_{i,y}(t) \rangle$$

- force acting on particle i at time t

$$\vec{F}_i(t) = \langle F_{i,x}(t), F_{i,y}(t) \rangle$$

N-body Problems: Newtonian Physics

Physical characteristics:



N-body Problems: Newtonian Physics

Motion and forces:

- velocity is the time derivative of position

$$\vec{v}_i(t) = \frac{d}{dt}\vec{p}_i(t)$$

- acceleration is the time derivative of velocity
- acceleration is the second time derivative of position

$$\vec{a}_i(t) = \frac{d}{dt}\vec{v}_i(t) = \frac{d^2}{dt^2}\vec{p}_i(t)$$

- Newton's Second Law of Motion

$$\vec{F}_i(t) = m \cdot \vec{a}_i(t) \qquad \vec{a}_i(t) = \frac{1}{m} \cdot \vec{F}_i(t)$$

N-body Problems: Newtonian Calculus

A particle's position at time $t + \delta$ is related to its position at time t according to a Taylor series expansion:

$$\vec{p}_i(t + \delta) = \vec{p}_i(t) + \delta \cdot \frac{d}{dt}\vec{p}_i(t) + \frac{\delta^2}{2} \cdot \frac{d^2}{dt^2}\vec{p}_i(t) + O(\delta^3)$$

N-body Problems: Newtonian Calculus

A particle's position at time $t + \delta$ is related to its position at time t according to a Taylor series expansion:

$$\vec{p}_i(t+\delta) = \vec{p}_i(t) + \delta \cdot \frac{d}{dt}\vec{p}_i(t) + \frac{\delta^2}{2} \cdot \frac{d^2}{dt^2}\vec{p}_i(t) + O(\delta^3)$$

If δ is small, then $O(\delta^3)$ is neglible
and a particle's future position may be approximated:

$$\vec{p}_i(t+\delta) \approx \vec{p}_i(t) + \delta \cdot \vec{v}_i(t) + \frac{\delta^2}{2} \cdot \vec{a}_i(t)$$

N-body Problems: Newtonian Calculus

A particle's velocity at time $t + \delta$ is related to its velocity at time t according to a Taylor series expansion:

$$\vec{v}_i(t + \delta) = \vec{v}_i(t) + \delta \cdot \frac{d}{dt}\vec{v}_i(t) + O(\delta^2)$$

N-body Problems: Newtonian Calculus

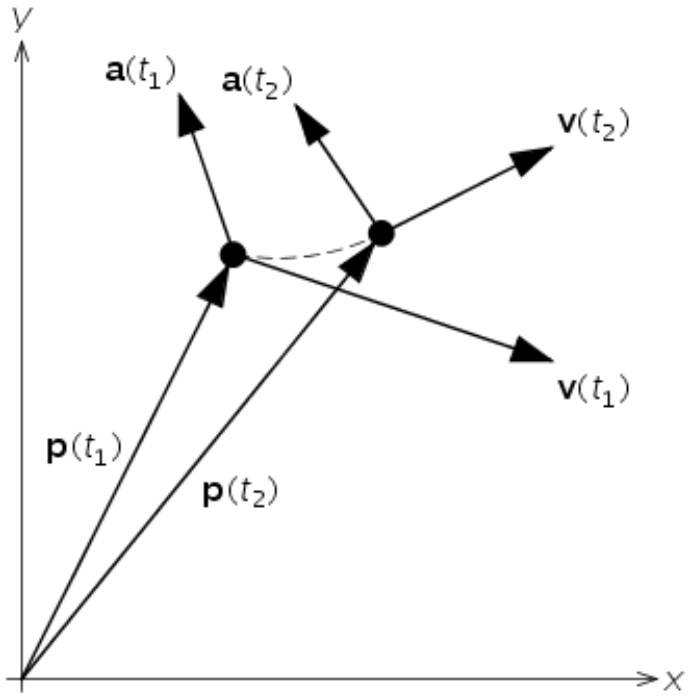
A particle's velocity at time $t + \delta$ is related to its velocity at time t according to a Taylor series expansion:

$$\vec{v}_i(t + \delta) = \vec{v}_i(t) + \delta \cdot \frac{d}{dt}\vec{v}_i(t) + O(\delta^2)$$

If δ is small, then $O(\delta^2)$ is neglible
and a particle's future velocity may be approximated:

$$\vec{v}_i(t + \delta) \approx \vec{v}_i(t) + \delta \cdot \vec{a}_i(t)$$

N-body Problems: Newtonian Calculus



$$t_2 = t_1 + \delta$$

$$\vec{p}_i(t_2) = \vec{p}_i(t_1) + \delta \cdot \vec{v}_i(t_1) + \frac{\delta^2}{2} \cdot \vec{a}_i(t_1)$$

$$\vec{v}_i(t_2) = \vec{v}_i(t_1) + \delta \cdot \vec{a}_i(t_1)$$

N-body Problems: Numerical Integration

Suppose we know the positions and velocities of n particles at time 0, and wish to predict their motion into the future.

N-body Problems: Numerical Integration

Suppose we know the positions and velocities of n particles at time 0, and wish to predict their motion into the future.

for $i = 0$ **to** $n - 1$
  $\vec{p}_i \leftarrow$ Position of particle i at time 0
  $\vec{v}_i \leftarrow$ Velocity of particle i at time 0
for $k = 1$ **to** *steps*
  for $i = 0$ **to** $n - 1$
    $\vec{F} \leftarrow$ Calculate force on particle i at time $(k\text{-}1)\delta$ from external fields and the positions and velocities of all particles
    $\vec{a}_i \leftarrow \frac{1}{m_i} \cdot \vec{F}$ *// Acceleration of particle i at time* $(k\text{-}1)\delta$
  for $i = 0$ **to** $n - 1$
    $\vec{p}_i \leftarrow \vec{p}_i + \delta\vec{v}_i + 0.5\delta^2\vec{a}_i$ *// Position of particle i at time* $k\delta$
    $\vec{v}_i \leftarrow \vec{v}_i + \delta\vec{a}_i$ *// Velocity of particle i at time* $k\delta$

N-body Problems: Numerical Integration

The previous is an example of a *numerical integration* algorithm:

- Solves the differential equation for $\vec{p}_i(t)$ via a sequence of discrete time-step calculations
- Uses ***steps*** *time steps*
- Uses a *second-order* integration algorithm (for positions)
  - Uses terms of order up to and including δ^2
- Incurs *truncation errors* due to the Taylor-series approximation
  - For accurate motion, requires a very small δ and a very large ***steps***

N-body Problems: Numerical Integration

The previous is an example of a *numerical integration* algorithm:

- Solves the differential equation for $\vec{p}_i(t)$ via a sequence of discrete time-step calculations
- Uses ***steps*** *time steps*
- Uses a *second-order* integration algorithm (for positions)
  - Uses terms of order up to and including δ^2
- Incurs *truncation errors* due to the Taylor-series approximation
  - For accurate motion, requires a very small δ and a very large ***steps***

Note: Only need the *last* positions and velocities of particles to calculate the *next* positions and velocities (like cellular automata algorithms).

N-body Problems: Numerical Integration

The previous is an example of a *numerical integration* algorithm:

- Solves the differential equation for $\vec{p}_i(t)$
  via a sequence of discrete time-step calculations
- Uses ***steps*** *time steps*
- Uses a *second-order* integration algorithm (for positions)
  - Uses terms of order up to and including δ^2
- Incurs *truncation errors* due to the Taylor-series approximation
  - For accurate motion, requires a very small δ and a very large ***steps***

Note: Only need the *last* positions and velocities of particles
to calculate the *next* positions and velocities (like cellular automata algorithms).

But, to visualize the particles' motion,
need to record particles' positions at different times.

N-body Problems: Numerical Integration

The previous is an example of a *numerical integration* algorithm:

- Solves the differential equation for $\vec{p}_i(t)$
  via a sequence of discrete time-step calculations
- Uses ***steps*** *time steps*
- Uses a *second-order* integration algorithm (for positions)
  - Uses terms of order up to and including δ^2
- Incurs *truncation errors* due to the Taylor-series approximation
  - For accurate motion, requires a very small δ and a very large ***steps***

Note: Only need the *last* positions and velocities of particles
to calculate the *next* positions and velocities (like cellular automata algorithms).

But, to visualize the particles' motion,
need to record particles' positions at different times.

With a very small δ, don't need to record particles' positions at *each* time;
only need to record a "snapshot" of positions after a number of time steps.

N-body Problems: Numerical Integration

Only need to record a "snapshot" of positions after a number of time steps.

N-body Problems: Numerical Integration

Only need to record a "snapshot" of positions after a number of time steps.

```
for i = 0 to n − 1
    p_i ← Position of particle i at time 0
    v_i ← Velocity of particle i at time 0
writeSnapshot(p)
for s = 1 to snaps
    for k = 1 to steps
        for i = 0 to n − 1
            F ← Calculate force on particle i at time ((s-1)steps+(k-1))δ
                from external fields and the positions and velocities of all particles
            a_i ← (1/m_i) · F   // Acceleration of particle i at time ((s-1)steps+(k-1))δ
        for i = 0 to n − 1
            p_i ← p_i + δv_i + 0.5δ²a_i   // Position of particle i at time ((s-1)steps+k)δ
            v_i ← v_i + δa_i   // Velocity of particle i at time ((s-1)steps+k)δ
        writeSnapshot(p)
```

N-body Problems: Numerical Integration

The previous is an example of a *numerical integration* algorithm:

- Solves the differential equation for $\vec{p}_i(t)$ via a sequence of discrete time-step calculations
- Uses $steps$ *time steps*
- Uses a *second-order* integration algorithm (for positions)
  - Uses terms of order up to and including δ^2
- Incurs *truncation errors* due to the Taylor-series approximation
  - For accurate motion, requires a very small δ and a very large s

Accuracy degrades due to truncation errors and floating-point arithmetic; need a mechanism to monitor accuracy.

N-body Problems: Numerical Integration

The previous is an example of a *numerical integration* algorithm:

- Solves the differential equation for $\vec{p}_i(t)$ via a sequence of discrete time-step calculations
- Uses ***steps*** *time steps*
- Uses a *second-order* integration algorithm (for positions)
  - Uses terms of order up to and including δ^2
- Incurs *truncation errors* due to the Taylor-series approximation
  - For accurate motion, requires a very small δ and a very large s

Accuracy degrades due to truncation errors and floating-point arithmetic; need a mechanism to monitor accuracy.

Keep track of a conserved quantity:

- momentum — Law of Conservation of Momentum
- energy — Law of Conservation of Energy

Such a quantity should remain constant.

N-body Problems: Numerical Integration

Keep track of a conserved quantity.

N-body Problems: Numerical Integration

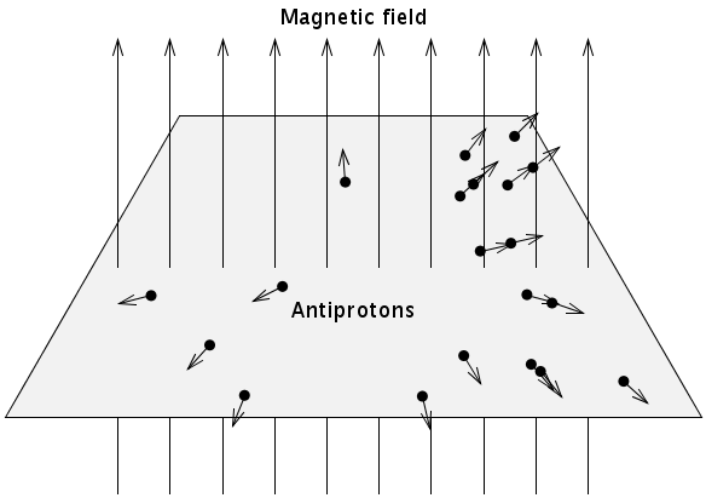
Keep track of a conserved quantity.

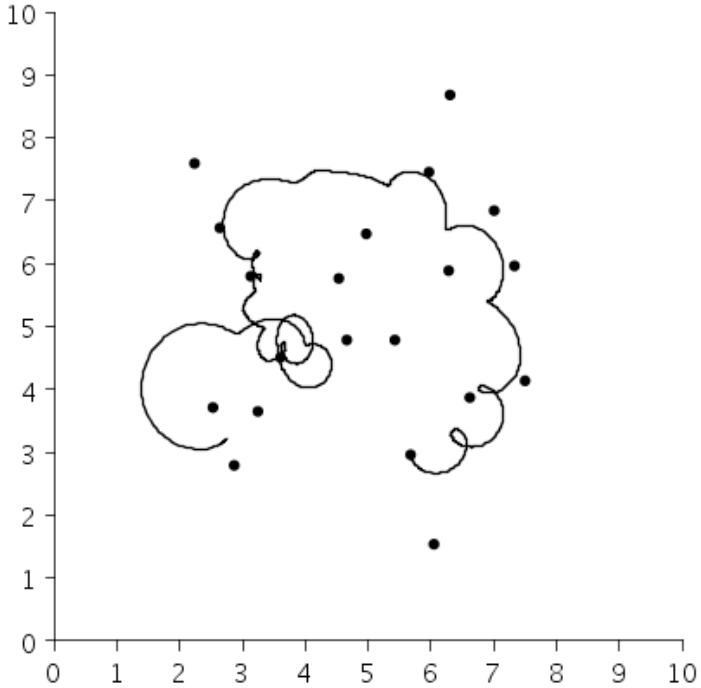
for $i = 0$ to $n - 1$
    $\vec{p}_i \leftarrow$ Position of particle i at time 0
    $\vec{v}_i \leftarrow$ Velocity of particle i at time 0
$cq \leftarrow$ Calculate conserved quantity at time 0 from positions and velocities of all particles
`writeSnapshot`$(\vec{p}, cq)$
for $s = 1$ to $snaps$
    for $k = 1$ to $steps$
        for $i = 0$ to $n - 1$
            $\vec{F} \leftarrow$ Calculate force on particle i at time $((s\text{-}1)steps+(k\text{-}1))\delta$ from external fields and the positions and velocities of all particles
            $\vec{a}_i \leftarrow \frac{1}{m_i} \cdot \vec{F}$ // *Acceleration of particle i at time* $((s\text{-}1)steps+(k\text{-}1))\delta$
        for $i = 0$ to $n - 1$
            $\vec{p}_i \leftarrow \vec{p}_i + \delta\vec{v}_i + 0.5\delta^2\vec{a}_i$ // *Position of particle i at time* $((s\text{-}1)steps+k)\delta$
            $\vec{v}_i \leftarrow \vec{v}_i + \delta\vec{a}_i$ // *Velocity of particle i at time* $((s\text{-}1)steps+k)\delta$
    $cq \leftarrow$ Calculate conserved quantity at time $((s\text{-}1)steps+k)\delta$ from positions and velocities of all particles
    `writeSnapshot`$(\vec{p}, cq)$

N-body Problems: Antiproton Motion

Electro-magnetic *N*-body problems:

- motion of charged particles (protons, antiprotons), due to mutual attraction/repulsion and magnetic fields





Antiproton Motion

Physical characteristics:

- charge of particle i

$$q_i$$

- force acting on particle i at time t

$$\vec{F}_i(t) = \sum_{j \neq i} \vec{F}_{e,i,j}(t) + \vec{F}_{m,i}(t)$$

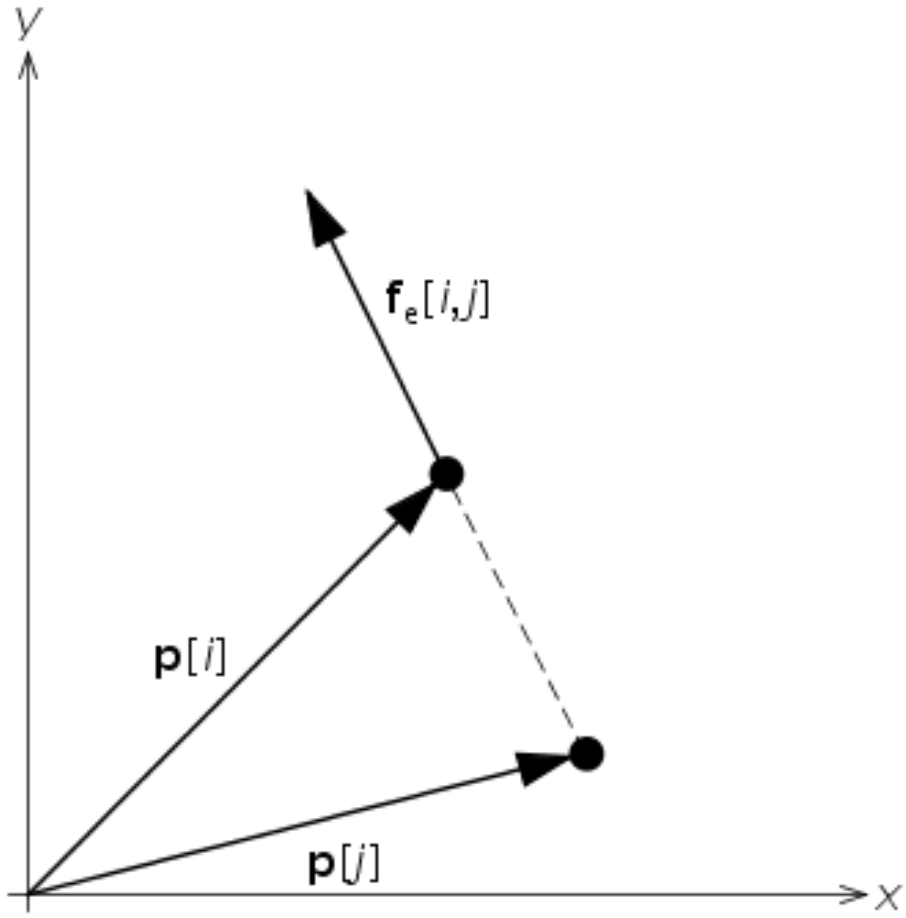
  - $\vec{F}_{e,i,j}(t)$: electrostatic force acting on particle i at time t from particle j
  - $\vec{F}_{m,i}(t)$: magnetic force acting on particle i at time t due to magnetic field

- total linear momentum of all particles at time t (conserved quantity)

$$\vec{P}(t) = \sum_i m_i \vec{v}_i(t)$$

Antiproton Motion: Coulomb's Law

$\vec{F}_{e,i,j}(t)$: electrostatic force acting on particle i at time t from particle j



Antiproton Motion: Coulomb's Law

$\vec{F}_{e,i,j}(t)$: electrostatic force acting on particle i at time t from particle j

$$\vec{F}_{e,i,j}(t) \propto \vec{p}_i(t) - \vec{p}_j(t)$$

$$\vec{F}_{e,i,j}(t) \propto \frac{\vec{p}_i(t) - \vec{p}_j(t)}{|\vec{p}_i(t) - \vec{p}_j(t)|}$$

Antiproton Motion: Coulomb's Law

$\vec{F}_{e,i,j}(t)$: electrostatic force acting on particle i at time t from particle j

$$\vec{F}_{e,i,j}(t) \propto \vec{p}_i(t) - \vec{p}_j(t)$$

$$\vec{F}_{e,i,j}(t) \propto \frac{\vec{p}_i(t) - \vec{p}_j(t)}{|\vec{p}_i(t) - \vec{p}_j(t)|}$$

$$\vec{F}_{e,i,j}(t) = k_e \frac{q_i q_j}{|\vec{p}_i(t) - \vec{p}_j(t)|^2} \cdot \frac{\vec{p}_i(t) - \vec{p}_j(t)}{|\vec{p}_i(t) - \vec{p}_j(t)|}$$

Antiproton Motion: Coulomb's Law

$\vec{F}_{e,i,j}(t)$: electrostatic force acting on particle i at time t from particle j

$$\vec{F}_{e,i,j}(t) \propto \vec{p}_i(t) - \vec{p}_j(t)$$

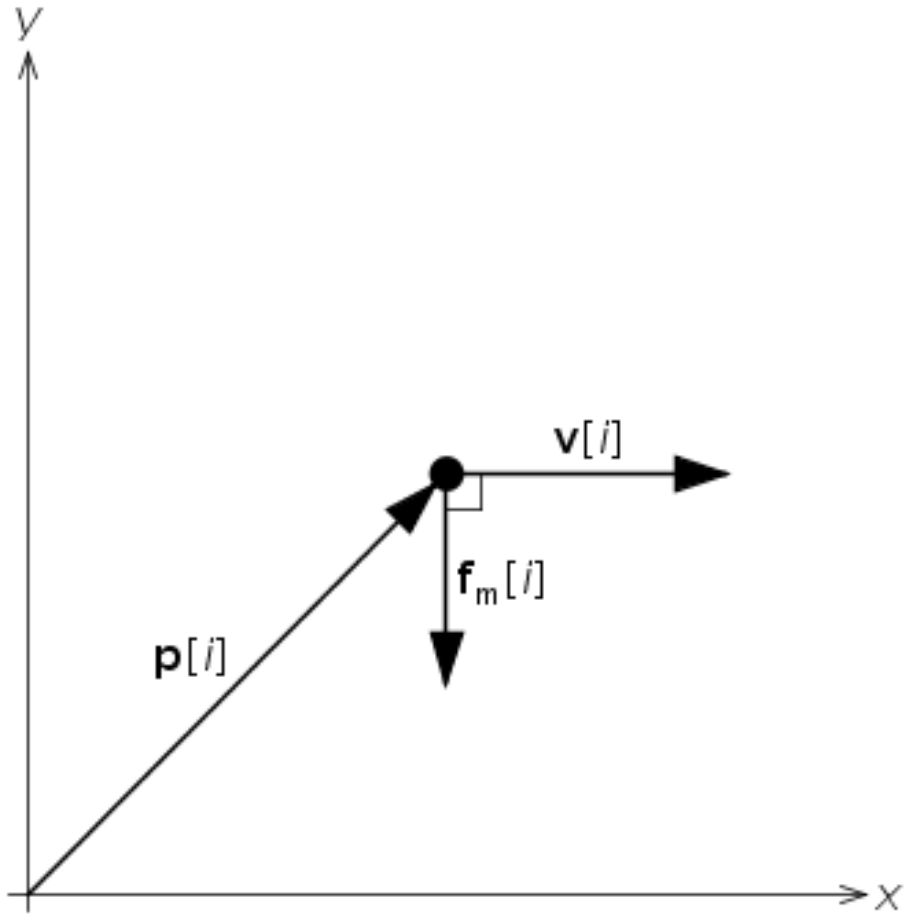
$$\vec{F}_{e,i,j}(t) \propto \frac{\vec{p}_i(t) - \vec{p}_j(t)}{|\vec{p}_i(t) - \vec{p}_j(t)|}$$

$$\vec{F}_{e,i,j}(t) = k_e \frac{q_i q_j}{|\vec{p}_i(t) - \vec{p}_j(t)|^2} \cdot \frac{\vec{p}_i(t) - \vec{p}_j(t)}{|\vec{p}_i(t) - \vec{p}_j(t)|}$$

$$\vec{F}_{e,i,j}(t) = \frac{k_e q_i q_j (\vec{p}_i(t) - \vec{p}_j(t))}{|\vec{p}_i(t) - \vec{p}_j(t)|^3}$$

Antiproton Motion: Lorentz Force

$\vec{F}_{m,i}(t)$: magnetic force acting on particle i at time t due to magnetic field



Antiproton Motion: Lorentz Force

$\vec{F}_{m,i}(t)$: magnetic force acting on particle i at time t due to magnetic field

$$\vec{F}_{m,i}(t) \propto \langle v_{i,y}(t), -v_{i,x}(t) \rangle$$

$$\vec{F}_{m,i}(t) \propto \frac{\langle v_{i,y}(t), -v_{i,x}(t) \rangle}{|\vec{v}_i(t)|}$$

Antiproton Motion: Lorentz Force

$\vec{F}_{m,i}(t)$: magnetic force acting on particle i at time t due to magnetic field

$$\vec{F}_{m,i}(t) \propto \langle v_{i,y}(t), -v_{i,x}(t) \rangle$$

$$\vec{F}_{m,i}(t) \propto \frac{\langle v_{i,y}(t), -v_{i,x}(t) \rangle}{|\vec{v}_i(t)|}$$

$$\vec{F}_{m,i}(t) = (q_i |\vec{v}_i(t)| B_m) \cdot \frac{\langle v_{i,y}(t), -v_{i,x}(t) \rangle}{|\vec{v}_i(t)|}$$

Antiproton Motion: Lorentz Force

$\vec{F}_{m,i}(t)$: magnetic force acting on particle i at time t due to magnetic field

$$\vec{F}_{m,i}(t) \propto \langle v_{i,y}(t), -v_{i,x}(t) \rangle$$

$$\vec{F}_{m,i}(t) \propto \frac{\langle v_{i,y}(t), -v_{i,x}(t) \rangle}{|\vec{v}_i(t)|}$$

$$\vec{F}_{m,i}(t) = (q_i |\vec{v}_i(t)| B_m) \cdot \frac{\langle v_{i,y}(t), -v_{i,x}(t) \rangle}{|\vec{v}_i(t)|}$$

$$\vec{F}_{m,i}(t) = q_i B_m \langle v_{i,y}(t), -v_{i,x}(t) \rangle$$

Antiproton Motion: Numerical Integration

```
for i = 0 to n − 1
    p⃗_i ← Position of particle i at time 0
    v⃗_i ← Velocity of particle i at time 0
P⃗ ← 0⃗
for i = 0 to n − 1
    P⃗ ← P⃗ + m_i v⃗_i
writeSnapshot(p⃗, P⃗)
for s = 1 to snaps
    for k = 1 to steps
        for i = 0 to n − 1
            F⃗ ← 0⃗
            for j = 0 to n − 1, j≠i
                d⃗ ← p⃗_i(t) − p⃗_j(t)
                F⃗ ← F⃗ + k_e q_i q_j · d⃗/|d⃗|^3
            F⃗ ← F⃗ + q_i B_m · ⟨v_{i,y}, −v_{i,x}⟩
            a⃗_i ← (1/m_i) · F⃗
        for i = 0 to n − 1
            p⃗_i ← p⃗_i + δv⃗_i + 0.5δ^2 a⃗_i
            v⃗_i ← v⃗_i + δa⃗_i
    P⃗ ← 0⃗
    for i = 0 to n − 1
        P⃗ ← P⃗ + m_i v⃗_i
    writeSnapshot(p⃗, P⃗)
```

Antiproton Motion: Numerical Integration

for $i = 0$ **to** $n - 1$
    $\vec{p}_i \leftarrow$ Position of particle i at time 0
    $\vec{v}_i \leftarrow$ Velocity of particle i at time 0
$\vec{P} \leftarrow \vec{0}$
for $i = 0$ **to** $n - 1$
    $\vec{P} \leftarrow \vec{P} + m_i \vec{v}_i$
`writeSnapshot(`$\vec{p}, \vec{P}$`)`
for $s = 1$ **to** *snaps*
    for $k = 1$ **to** *steps*
        for $i = 0$ **to** $n - 1$
            $\vec{a}_i \leftarrow \vec{0}$
            for $j = 0$ **to** $n - 1$, $j \neq i$
                $\vec{d} \leftarrow \vec{p}_i(t) - \vec{p}_j(t)$
                $\vec{a}_i \leftarrow \vec{a}_i + \frac{1}{m_i} \cdot k_e q_i q_j \cdot \vec{d} / |\vec{d}|^3$
            $\vec{a}_i \leftarrow \vec{a}_i + \frac{1}{m_i} \cdot q_i B_m \cdot \langle v_{i,y}, -v_{i,x} \rangle$
        for $i = 0$ **to** $n - 1$
            $\vec{p}_i \leftarrow \vec{p}_i + \delta \vec{v}_i + 0.5\delta^2 \vec{a}_i$
            $\vec{v}_i \leftarrow \vec{v}_i + \delta \vec{a}_i$
    $\vec{P} \leftarrow \vec{0}$
    for $i = 0$ **to** $n - 1$
        $\vec{P} \leftarrow \vec{P} + m_i \vec{v}_i$
    `writeSnapshot(`$\vec{p}, \vec{P}$`)`

Calculate acceleration directly.

Antiproton Motion: Numerical Integration

```
for i = 0 to n − 1
    p⃗_i ← Position of particle i at time 0
    v⃗_i ← Velocity of particle i at time 0
P⃗ ← 0⃗
for i = 0 to n − 1
    P⃗ ← P⃗ + 1 · v⃗_i
writeSnapshot(p⃗, P⃗)
for s = 1 to snaps
    for k = 1 to steps
        for i = 0 to n − 1
            a⃗_i ← 0⃗
            for j = 0 to n − 1, j≠i
                d⃗ ← p⃗_i(t) − p⃗_j(t)
                a⃗_i ← a⃗_i + Q · d⃗/|d⃗|^3
            a⃗_i ← a⃗_i + B · ⟨v_{i,y}, −v_{i,x}⟩
        for i = 0 to n − 1
            p⃗_i ← p⃗_i + δv⃗_i + 0.5δ^2 a⃗_i
            v⃗_i ← v⃗_i + δa⃗_i
    P⃗ ← 0⃗
    for i = 0 to n − 1
        P⃗ ← P⃗ + 1 · v⃗_i
    writeSnapshot(p⃗, P⃗)
```

Assume all particles have unit mass and unit charge (by adjusting units and introducing constants Q and B).

Antiproton Motion: Numerical Integration

$$
\begin{array}{l}
\textbf{for } i = 0 \textbf{ to } n-1 \\
\quad \vec{p}_i \leftarrow \langle \texttt{random}(R/4, 3R/4), \texttt{random}(R/4, 3R/4) \rangle \\
\quad \vec{v}_i \leftarrow \vec{0} \\
\vec{P} \leftarrow \vec{0} \\
\texttt{writeSnapshot}(\vec{p}, \vec{P}) \\
\textbf{for } s = 1 \textbf{ to } \textit{snaps} \\
\quad \textbf{for } k = 1 \textbf{ to } \textit{steps} \\
\quad\quad \textbf{for } i = 0 \textbf{ to } n-1 \\
\quad\quad\quad \vec{a}_i \leftarrow \vec{0} \\
\quad\quad\quad \textbf{for } j = 0 \textbf{ to } n-1, \ j \neq i \\
\quad\quad\quad\quad \vec{d} \leftarrow \vec{p}_i(t) - \vec{p}_j(t) \\
\quad\quad\quad\quad \vec{a}_i \leftarrow \vec{a}_i + Q \cdot \vec{d} / |\vec{d}|^3 \\
\quad\quad\quad \vec{a}_i \leftarrow \vec{a}_i + B \cdot \langle v_{i,y}, -v_{i,x} \rangle \\
\quad\quad \textbf{for } i = 0 \textbf{ to } n-1 \\
\quad\quad\quad \vec{p}_i \leftarrow \vec{p}_i + \delta \vec{v}_i + 0.5 \delta^2 \vec{a}_i \\
\quad\quad\quad \vec{v}_i \leftarrow \vec{v}_i + \delta \vec{a}_i \\
\quad \vec{P} \leftarrow \vec{0} \\
\quad \textbf{for } i = 0 \textbf{ to } n-1 \\
\quad\quad \vec{P} \leftarrow \vec{P} + 1 \cdot \vec{v}_i \\
\quad \texttt{writeSnapshot}(\vec{p}, \vec{P})
\end{array}
$$

Initial position is random (near center of the square $\langle 0, 0 \rangle$ to $\langle R, R \rangle$).
Initial velocity is $\vec{0}$.

Antiproton Motion: Numerical Integration

$$
\begin{array}{l}
\textbf{for } i = 0 \textbf{ to } n-1 \\
\quad \vec{p}_i \leftarrow \langle \texttt{random}(R/4, 3R/4), \texttt{random}(R/4, 3R/4) \rangle \\
\quad \vec{v}_i \leftarrow \vec{0} \\
\quad \vec{a}_i \leftarrow \vec{0} \\
\vec{P} \leftarrow \vec{0} \\
\texttt{writeSnapshot}(\vec{p}, \vec{P}) \\
\textbf{for } s = 1 \textbf{ to } \textit{snaps} \\
\quad \textbf{for } k = 1 \textbf{ to } \textit{steps} \\
\quad\quad \textbf{for } i = 0 \textbf{ to } n-1 \\
\quad\quad\quad \textbf{for } j = 0 \textbf{ to } n-1, \ j \neq i \\
\quad\quad\quad\quad \vec{d} \leftarrow \vec{p}_i(t) - \vec{p}_j(t) \\
\quad\quad\quad\quad \vec{a}_i \leftarrow \vec{a}_i + Q \cdot \vec{d} / |\vec{d}|^3 \\
\quad\quad \textbf{for } i = 0 \textbf{ to } n-1 \\
\quad\quad\quad \vec{a}_i \leftarrow \vec{a}_i + B \cdot \langle v_{i,y}, -v_{i,x} \rangle \\
\quad\quad\quad \vec{p}_i \leftarrow \vec{p}_i + \delta \vec{v}_i + 0.5 \delta^2 \vec{a}_i \\
\quad\quad\quad \vec{v}_i \leftarrow \vec{v}_i + \delta \vec{a}_i \\
\quad\quad\quad \vec{a}_i \leftarrow \vec{0} \\
\quad \vec{P} \leftarrow \vec{0} \\
\quad \textbf{for } i = 0 \textbf{ to } n-1 \\
\quad\quad \vec{P} \leftarrow \vec{P} + 1 \cdot \vec{v}_i \\
\quad \texttt{writeSnapshot}(\vec{p}, \vec{P})
\end{array}
$$

Rearrange acceleration computation.

AntiprotonSeq.java

code/AntiprotonSeq.java

- *seed* — PRNG seed for particles' initial positions
- *R* — side of the square for particles' initial positions
- *dt* — size of the time step, δ
- *steps* — number of time steps in each snapshot
- *snaps* — numer of snapshots
- *n* — number of particles
- *outfile* — output file name

- `edu.rit.clu.antimatter.AntiprotonFile` — handles file I/O of snapshots
  - `edu.rit.clu.antimatter.AntiprotonPlot` — produces visualizations
- `edu.rit.vector.Vector2D` — vector arithmetic

Antiproton Motion on a Cluster

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

```
for i = 0 to n − 1
    p_i ← ⟨random(R/4, 3R/4), random(R/4, 3R/4)⟩
    v_i ← 0
    a_i ← 0
P ← 0
writeSnapshot(p, P)
for s = 1 to snaps
    for k = 1 to steps
        for i = 0 to n − 1
            for j = 0 to n − 1, j≠i
                d ← p_i(t) − p_j(t)
                a_i ← a_i + Q · d/|d|^3
        for i = 0 to n − 1
            a_i ← a_i + B · ⟨v_{i,y}, −v_{i,x}⟩
            p_i ← p_i + δ v_i + 0.5 δ^2 a_i
            v_i ← v_i + δ a_i
            a_i ← 0
    P ← 0
    for i = 0 to n − 1
        P ← P + 1 · v_i
    writeSnapshot(p, P)
```

Antiproton Motion on a Cluster

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

```
for i = 0 to n − 1
    p⃗_i ← ⟨random(R/4, 3R/4), random(R/4, 3R/4)⟩
    v⃗_i ← 0⃗
    a⃗_i ← 0⃗
P⃗ ← 0⃗
writeSnapshot(p⃗, P⃗)
for s = 1 to snaps
    for k = 1 to steps
        for i = 0 to n − 1
            for j = 0 to n − 1, j≠i
                d⃗ ← p⃗_i(t) − p⃗_j(t)
                a⃗_i ← a⃗_i + Q · d⃗/|d⃗|^3
        for i = 0 to n − 1
            a⃗_i ← a⃗_i + B · ⟨v_{i,y}, −v_{i,x}⟩
            p⃗_i ← p⃗_i + δv⃗_i + 0.5δ^2 a⃗_i
            v⃗_i ← v⃗_i + δa⃗_i
            a⃗_i ← 0⃗
    P⃗ ← 0⃗
    for i = 0 to n − 1
        P⃗ ← P⃗ + 1 · v⃗_i
    writeSnapshot(p⃗, P⃗)
```

Partition the n particles among the K processes;
each process responsible for n/K particles (position, velocity, acceleration).

Antiproton Motion on a Cluster

Partition the n particles among the K processes;
each process responsible for n/K particles (position, velocity, acceleration).

$lb, ub \leftarrow$ `range`$(0, n-1, K, rank)$
for $i = lb$ **to** ub
  $\vec{p}_i \leftarrow \langle$`random`$(R/4, 3R/4)$, `random`$(R/4, 3R/4)\rangle$
  $\vec{v}_i \leftarrow \vec{0}$
  $\vec{a}_i \leftarrow \vec{0}$
$\vec{P} \leftarrow \vec{0}$
`writeSnapshot`$(\vec{p}, \vec{P})$
for $s = 1$ **to** $snaps$
  for $k = 1$ **to** $steps$
    for $i = lb$ **to** ub
      for $j = 0$ **to** $n-1$, $j \neq i$
        $\vec{d} \leftarrow \vec{p}_i(t) - \vec{p}_j(t)$
        $\vec{a}_i \leftarrow \vec{a}_i + Q \cdot \vec{d}/|\vec{d}|^3$
    for $i = lb$ **to** ub
      $\vec{a}_i \leftarrow \vec{a}_i + B \cdot \langle v_{i,y}, -v_{i,x}\rangle$
      $\vec{p}_i \leftarrow \vec{p}_i + \delta\vec{v}_i + 0.5\delta^2\vec{a}_i$
      $\vec{v}_i \leftarrow \vec{v}_i + \delta\vec{a}_i$
      $\vec{a}_i \leftarrow \vec{0}$
  $\vec{P} \leftarrow \vec{0}$
  for $i = lb$ **to** ub
    $\vec{P} \leftarrow \vec{P} + 1 \cdot \vec{v}_i$
  `writeSnapshot`$(\vec{p}, \vec{P})$

Antiproton Motion on a Cluster

Partition the n particles among the K processes;
each process responsible for n/K particles (position, velocity, acceleration).

$lb, ub \leftarrow$ `range`$(0, n-1, K, rank)$
for $i = lb$ **to** ub
  $\vec{p}_i \leftarrow \langle$`random`$(R/4, 3R/4)$, `random`$(R/4, 3R/4)\rangle$
  $\vec{v}_i \leftarrow \vec{0}$
  $\vec{a}_i \leftarrow \vec{0}$
$\vec{P} \leftarrow \vec{0}$
`writeSnapshot`$(\vec{p}, \vec{P})$
for $s = 1$ **to** $snaps$
  for $k = 1$ **to** $steps$
    for $i = lb$ **to** ub
      for $j = 0$ **to** $n-1$, $j \neq i$
        $\vec{d} \leftarrow \vec{p}_i(t) - \vec{p}_j(t)$
        $\vec{a}_i \leftarrow \vec{a}_i + Q \cdot \vec{d}/|\vec{d}|^3$
    for $i = lb$ **to** ub
      $\vec{a}_i \leftarrow \vec{a}_i + B \cdot \langle v_{i,y}, -v_{i,x}\rangle$
      $\vec{p}_i \leftarrow \vec{p}_i + \delta\vec{v}_i + 0.5\delta^2\vec{a}_i$
      $\vec{v}_i \leftarrow \vec{v}_i + \delta\vec{a}_i$
      $\vec{a}_i \leftarrow \vec{0}$
  $\vec{P} \leftarrow \vec{0}$
  for $i = lb$ **to** ub
    $\vec{P} \leftarrow \vec{P} + 1 \cdot \vec{v}_i$
  `writeSnapshot`$(\vec{p}, \vec{P})$

Note: use parallel output files pattern.

Antiproton Motion on a Cluster

Partition the n particles among the K processes;
each process responsible for n/K particles (position, velocity, acceleration).

$$
\begin{array}{l}
\textbf{for } i = lb \textbf{ to } ub \\
\quad \textbf{for } j = 0 \textbf{ to } n-1, \;\; j \neq i \\
\quad\quad \vec{d} \leftarrow \vec{p}_i(t) - \vec{p}_j(t) \\
\quad\quad \vec{a}_i \leftarrow \vec{a}_i + Q \cdot \vec{d}/|\vec{d}|^3 \\
\textbf{for } i = lb \textbf{ to } ub \\
\quad \vec{a}_i \leftarrow \vec{a}_i + B \cdot \langle v_{i,y}, -v_{i,x} \rangle \\
\quad \vec{p}_i \leftarrow \vec{p}_i + \delta \vec{v}_i + 0.5\delta^2 \vec{a}_i \\
\quad \vec{v}_i \leftarrow \vec{v}_i + \delta \vec{a}_i \\
\quad \vec{a}_i \leftarrow \vec{0}
\end{array}
$$

What data is accessed by each process on each iteration?

Antiproton Motion on a Cluster

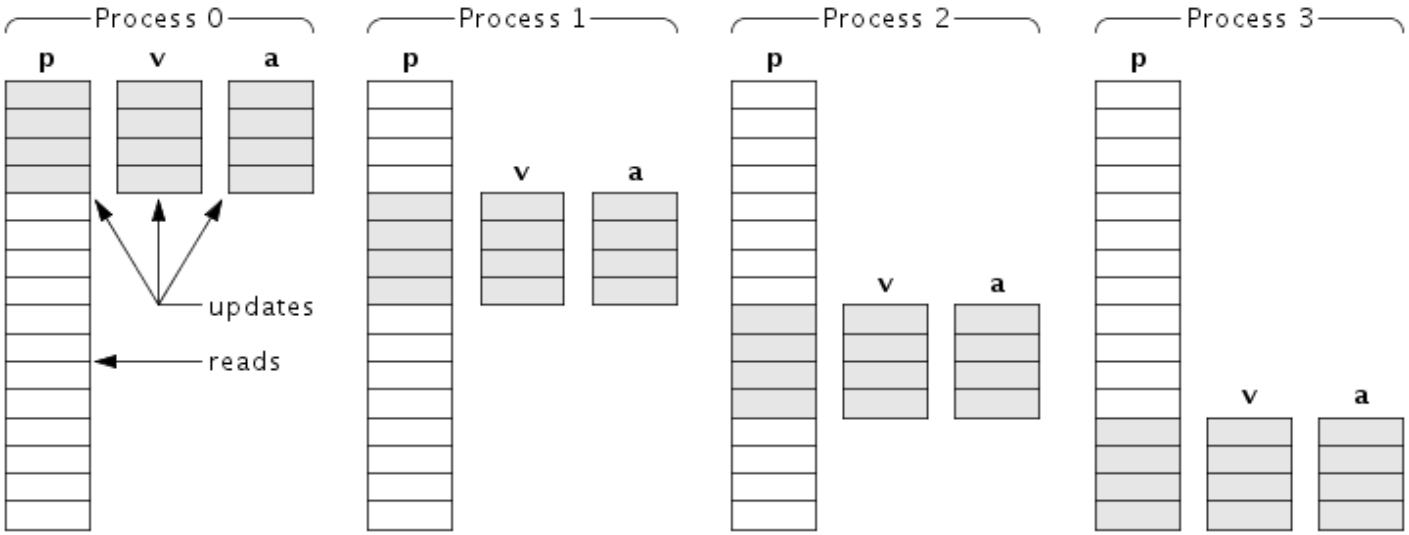
Partition the n particles among the K processes;
each process responsible for n/K particles (position, velocity, acceleration).

$$
\begin{array}{l}
\textbf{for } i = lb \textbf{ to } ub \\
\quad \textbf{for } j = 0 \textbf{ to } n-1, \;\; j \neq i \\
\quad\quad \vec{d} \leftarrow \vec{p}_i(t) - \vec{p}_j(t) \\
\quad\quad \vec{a}_i \leftarrow \vec{a}_i + Q \cdot \vec{d}/|\vec{d}|^3 \\
\textbf{for } i = lb \textbf{ to } ub \\
\quad \vec{a}_i \leftarrow \vec{a}_i + B \cdot \langle v_{i,y}, -v_{i,x} \rangle \\
\quad \vec{p}_i \leftarrow \vec{p}_i + \delta \vec{v}_i + 0.5\delta^2 \vec{a}_i \\
\quad \vec{v}_i \leftarrow \vec{v}_i + \delta \vec{a}_i \\
\quad \vec{a}_i \leftarrow \vec{0}
\end{array}
$$

What data is accessed by each process on each iteration?

Every process updates its slice of position, velocity, and acceleration arrays.
But, every process reads the entire position array.

Antiproton Motion on a Cluster



Each process need only allocate storage
for its slice of velocity and acceleration arrays,
but must allocate storage for entire position array.

Antiproton Motion on a Cluster

Partition the n particles among the K processes;
each process responsible for n/K particles (position, velocity, acceleration).

$$
\begin{array}{l}
\textbf{for } i = lb \textbf{ to } ub \\
\quad \textbf{for } j = 0 \textbf{ to } n-1, \;\; j \neq i \\
\quad\quad \vec{d} \leftarrow \vec{p}_i(t) - \vec{p}_j(t) \\
\quad\quad \vec{a}_i \leftarrow \vec{a}_i + Q \cdot \vec{d}/|\vec{d}|^3 \\
\textbf{for } i = lb \textbf{ to } ub \\
\quad \vec{a}_i \leftarrow \vec{a}_i + B \cdot \langle v_{i,y}, -v_{i,x} \rangle \\
\quad \vec{p}_i \leftarrow \vec{p}_i + \delta \vec{v}_i + 0.5\delta^2 \vec{a}_i \\
\quad \vec{v}_i \leftarrow \vec{v}_i + \delta \vec{a}_i \\
\quad \vec{a}_i \leftarrow \vec{0}
\end{array}
$$

Antiproton Motion on a Cluster

Partition the n particles among the K processes;
each process responsible for n/K particles (position, velocity, acceleration).

$$
\begin{array}{l}
\textbf{for } i = lb \textbf{ to } ub \\
\quad \textbf{for } j = 0 \textbf{ to } n-1, \;\; j \neq i \\
\quad\quad \vec{d} \leftarrow \vec{p}_i(t) - \vec{p}_j(t) \\
\quad\quad \vec{a}_i \leftarrow \vec{a}_i + Q \cdot \vec{d}/|\vec{d}|^3 \\
\textbf{for } i = lb \textbf{ to } ub \\
\quad \vec{a}_i \leftarrow \vec{a}_i + B \cdot \langle v_{i,y}, -v_{i,x} \rangle \\
\quad \vec{p}_i \leftarrow \vec{p}_i + \delta \vec{v}_i + 0.5\delta^2 \vec{a}_i \\
\quad \vec{v}_i \leftarrow \vec{v}_i + \delta \vec{a}_i \\
\quad \vec{a}_i \leftarrow \vec{0}
\end{array}
$$

Every process updates its slice of position, velocity, and acceleration arrays.
But, every process reads the entire position array.

After updating its slice of position and velocity,
but before calculating its next slice of acceleration,
every process must communicate its slice of position to every other process.

Which collective communication operation?

Antiproton Motion on a Cluster

Partition the n particles among the K processes;
each process responsible for n/K particles (position, velocity, acceleration).

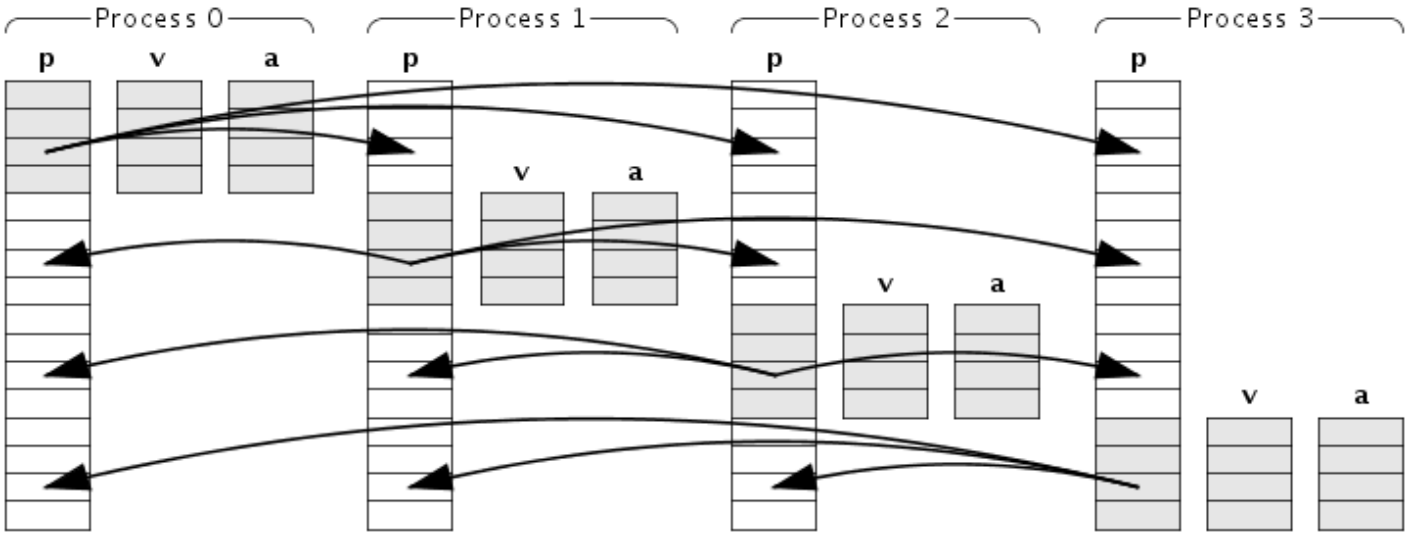
$$
\begin{array}{l}
\textbf{for } i = lb \textbf{ to } ub \\
\quad \textbf{for } j = 0 \textbf{ to } n-1, \;\; j \neq i \\
\quad\quad \vec{d} \leftarrow \vec{p}_i(t) - \vec{p}_j(t) \\
\quad\quad \vec{a}_i \leftarrow \vec{a}_i + Q \cdot \vec{d}/|\vec{d}|^3 \\
\textbf{for } i = lb \textbf{ to } ub \\
\quad \vec{a}_i \leftarrow \vec{a}_i + B \cdot \langle v_{i,y}, -v_{i,x} \rangle \\
\quad \vec{p}_i \leftarrow \vec{p}_i + \delta \vec{v}_i + 0.5\delta^2 \vec{a}_i \\
\quad \vec{v}_i \leftarrow \vec{v}_i + \delta \vec{a}_i \\
\quad \vec{a}_i \leftarrow \vec{0}
\end{array}
$$

Every process updates its slice of position, velocity, and acceleration arrays.
But, every process reads the entire position array.

After updating its slice of position and velocity,
but before calculating its next slice of acceleration,
every process must communicate its slice of position to every other process.

Which collective communication operation? all-gather

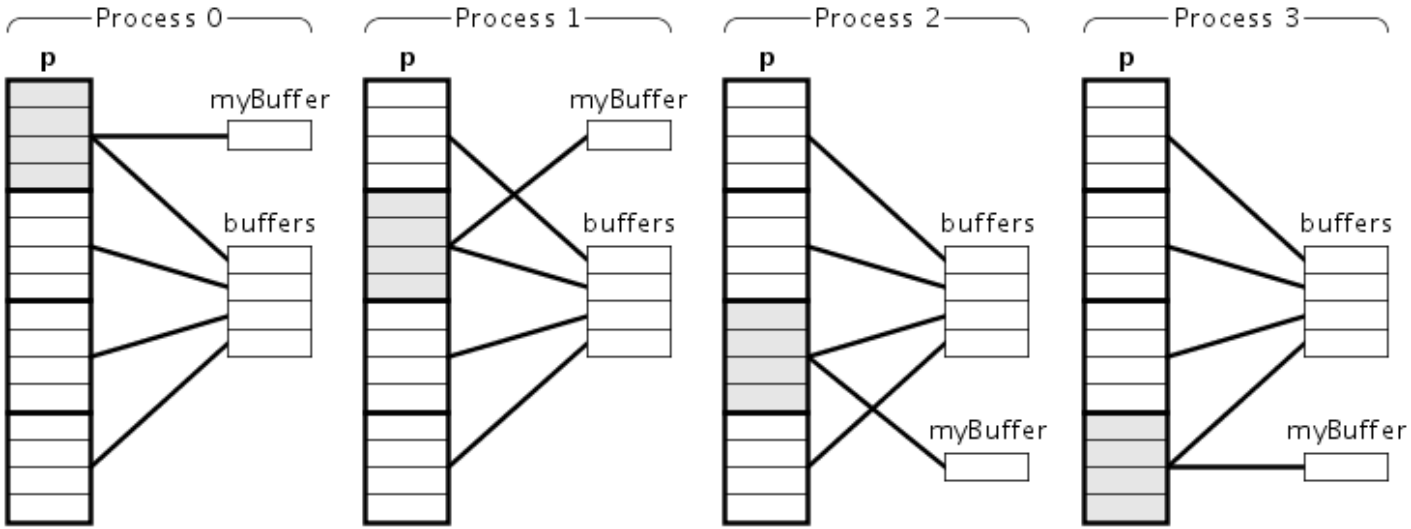
Antiproton Motion on a Cluster



After updating its slice of position and velocity,
but before calculating its next slice of acceleration,
every process must communicate its slice of position to every other process.

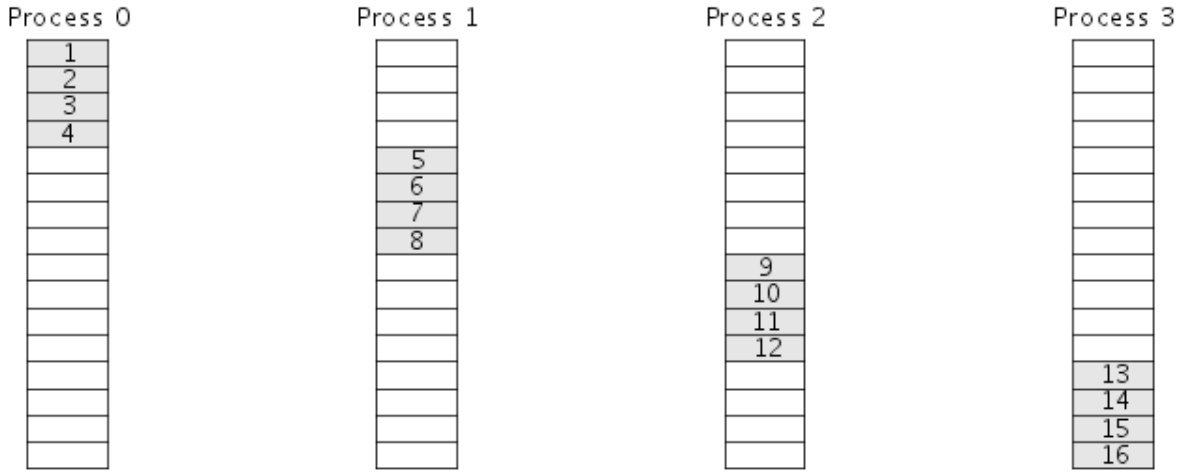
Also acts as a synchronization point to enforce the sequential dependency from one time step to the next.

Antiproton Motion on a Cluster



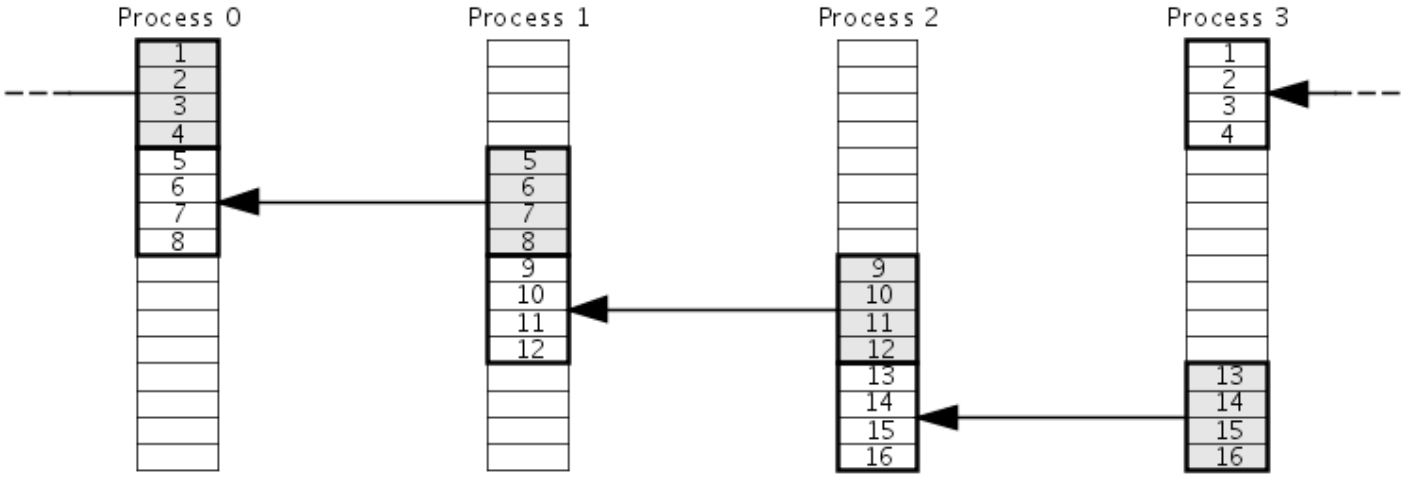

```
world.allGather (myBuffer, buffers);
```

All-Gather Implementation and Message Time Model



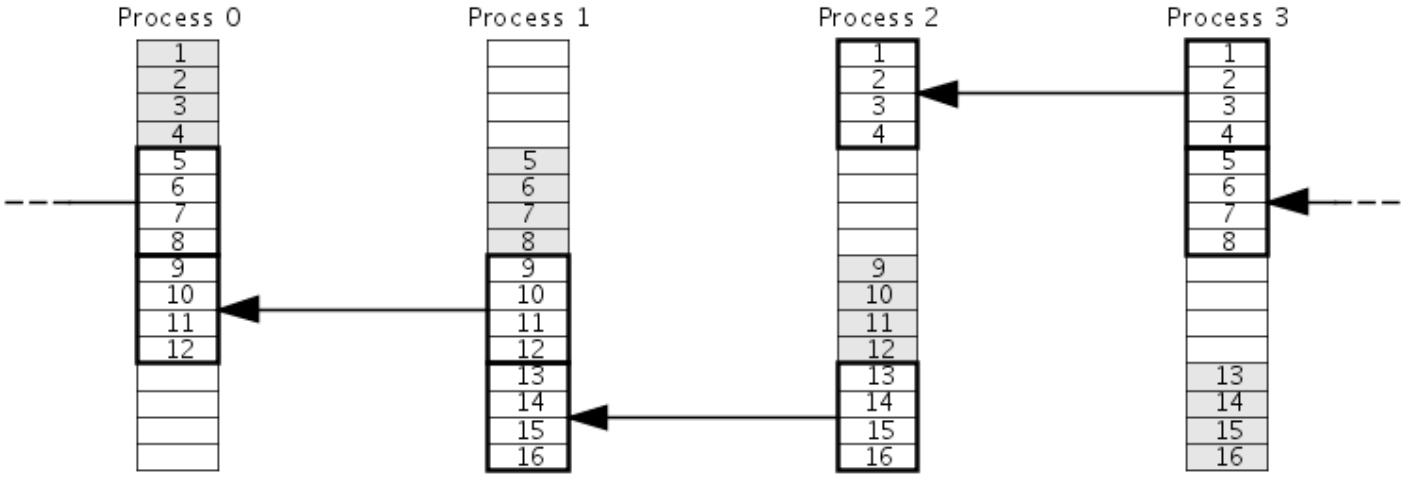
Each node simultaneously sends one data buffer to its predecessor and receives another data buffer from its successor.
First round, sends its own data buffer;
later rounds, sends last received data buffer.

All-Gather Implementation and Message Time Model



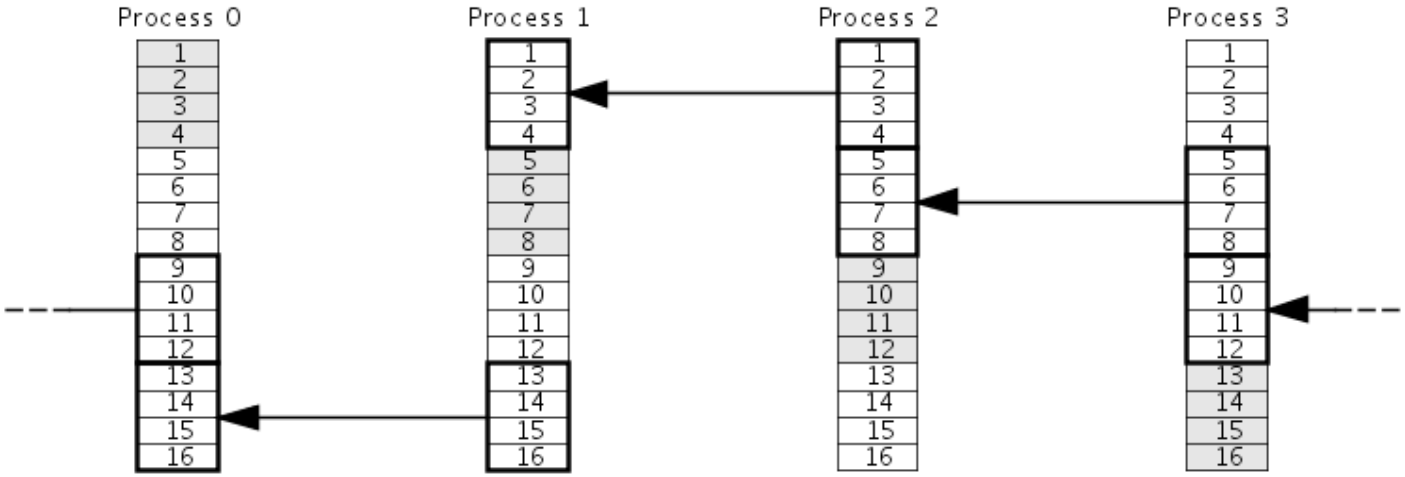
Each node simultaneously sends one data buffer to its predecessor and receives another data buffer from its successor.
First round, sends its own data buffer;
later rounds, sends last received data buffer.

All-Gather Implementation and Message Time Model



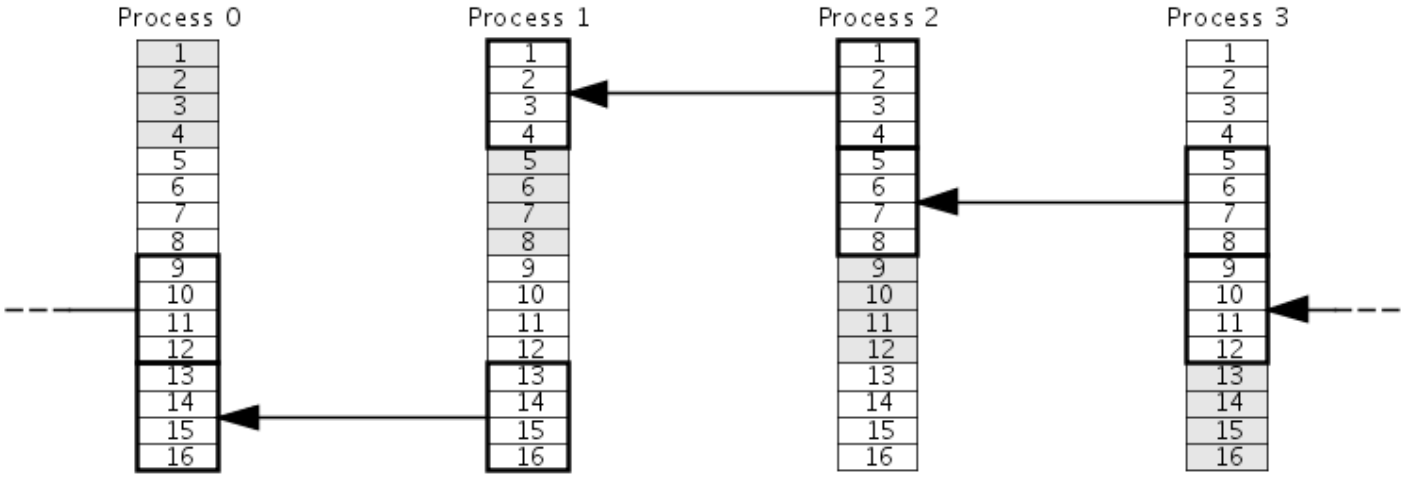
Each node simultaneously sends one data buffer to its predecessor and receives another data buffer from its successor.
First round, sends its own data buffer;
later rounds, sends last received data buffer.

All-Gather Implementation and Message Time Model



Each node simultaneously sends one data buffer to its predecessor and receives another data buffer from its successor.
First round, sends its own data buffer;
later rounds, sends last received data buffer.

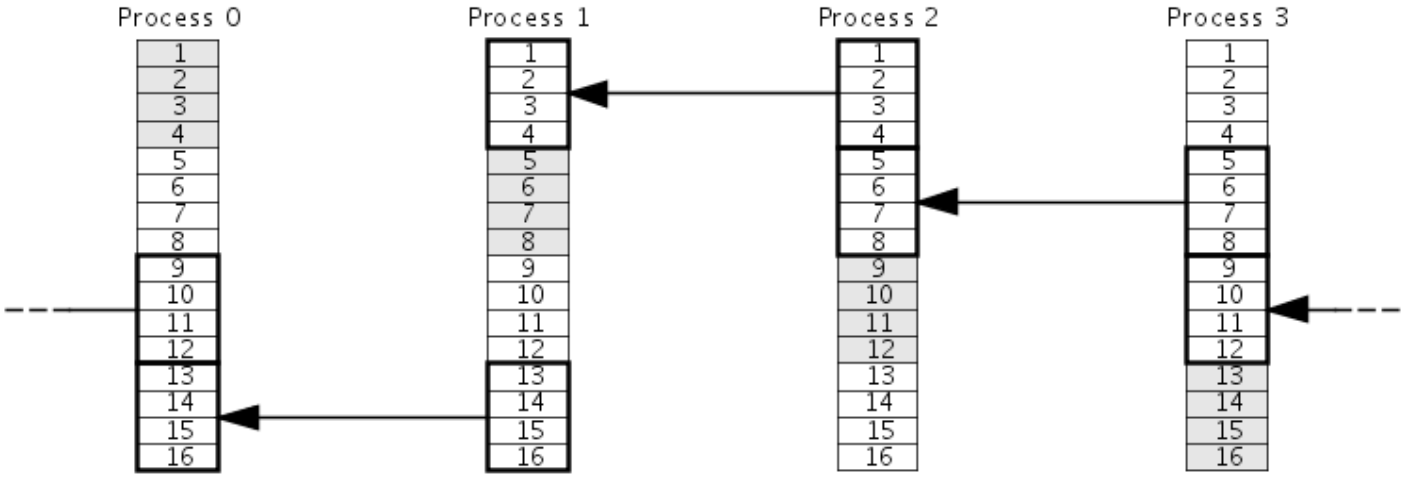
All-Gather Implementation and Message Time Model



Each node simultaneously sends one data buffer to its predecessor and receives another data buffer from its successor.
First round, sends its own data buffer;
later rounds, sends last received data buffer.

$$T_{\text{all-gather}}(b, K) =$$

All-Gather Implementation and Message Time Model



Each node simultaneously sends one data buffer to its predecessor and receives another data buffer from its successor.
First round, sends its own data buffer;
later rounds, sends last received data buffer.

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

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

AntiprotonClu.java

code/AntiprotonClu.java

AntiprotonClu.java

code/AntiprotonClu.java

Before looking at the running-time measurements for AntiprotonClu, derive a model to predict the running time.

What will we need?

- a calculation time model
- a communication time model

Calculation Time Model

Antiproton motion algorithm is $O(sn^2)$,
where n is the number of particles
and s is total number of steps ($s = snaps \cdot steps$).

$$T^{\bar{p}}_{\mathrm{calc}}(s, n, 1) = asn^2$$

$$T^{\bar{p}}_{\mathrm{calc}}(s, n, K) = asn^2 \frac{1}{K}$$

$$T^{\bar{p}}_{\mathrm{calc}}(s, n, 1) = 2.04 \times 10^{-8} sn^2$$

$$T^{\bar{p}}_{\mathrm{calc}}(s, n, K) = 2.04 \times 10^{-8} sn^2 \frac{1}{K}$$

Communication Time Model

- How many all-gathers are performed?
- How many bits in each all-gather message?

Communication Time Model

- How many all-gathers are performed? s
- How many bits in each all-gather message? $128n\frac{1}{K}$

Communication Time Model

- How many all-gathers are performed? s
- How many bits in each all-gather message? $128n\frac{1}{K}$

$$T^{\bar{p}}_{\text{comm}}(s, n, K) = sT_{\text{all-gather}}(128n\tfrac{1}{K}, K)$$

$$T^{\bar{p}}_{\text{comm}}(s, n, K) = s(L + \tfrac{1}{B}128n\tfrac{1}{K})(K - 1)$$

$$T^{\bar{p}}_{\text{comm}}(s, n, K) = s(2.08\times10^{-4} + 1.37\times10^{-7}n\tfrac{1}{K})(K - 1)$$

Running Time Model

$$T^{\bar{p}}(s,n,K) = T^{\bar{p}}_{\text{calc}}(s,n,K) + T^{\bar{p}}_{\text{comm}}(s,n,K)$$

$$T^{\bar{p}}(s,n,K) = asn^2\frac{1}{K} + s(L + \frac{1}{B}128n\frac{1}{K})(K-1)$$

$$T^{\bar{p}}(s,n,K) = 2.04\times10^{-8}sn^2\frac{1}{K} + s(2.08\times10^{-4} + 1.37\times10^{-7}n\frac{1}{K})(K-1)$$

Running Time Model

$$T^{\bar{p}}(s,n,K) = T^{\bar{p}}_{\text{calc}}(s,n,K) + T^{\bar{p}}_{\text{comm}}(s,n,K)$$

$$T^{\bar{p}}(s,n,K) = asn^2\frac{1}{K} + s(L + \frac{1}{B}128n\frac{1}{K})(K-1)$$

$$T^{\bar{p}}(s,n,K) = 2.04\times10^{-8}sn^2\frac{1}{K} + s(2.08\times10^{-4} + 1.37\times10^{-7}n\frac{1}{K})(K-1)$$

Classic cluster parallel program problem:

- calculation time decreases as a function of K
- communication time increases as a function of K

Running Time Model

$$T^{\bar{p}}(s, n, K) = T^{\bar{p}}_{\text{calc}}(s, n, K) + T^{\bar{p}}_{\text{comm}}(s, n, K)$$

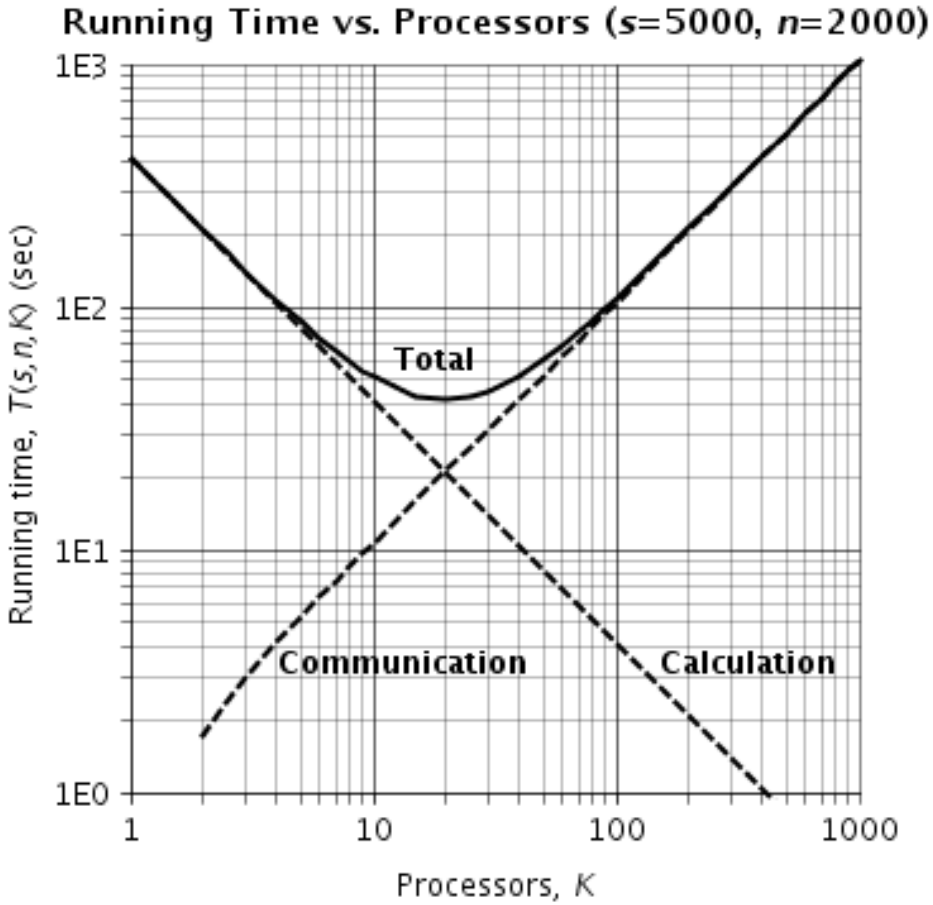
$$T^{\bar{p}}(s, n, K) = asn^2\frac{1}{K} + s(L + \frac{1}{B}128n\frac{1}{K})(K-1)$$

$$T^{\bar{p}}(s, n, K) = 2.04\times10^{-8}sn^2\frac{1}{K} + s(2.08\times10^{-4} + 1.37\times10^{-7}n\frac{1}{K})(K-1)$$

Classic cluster parallel program problem:

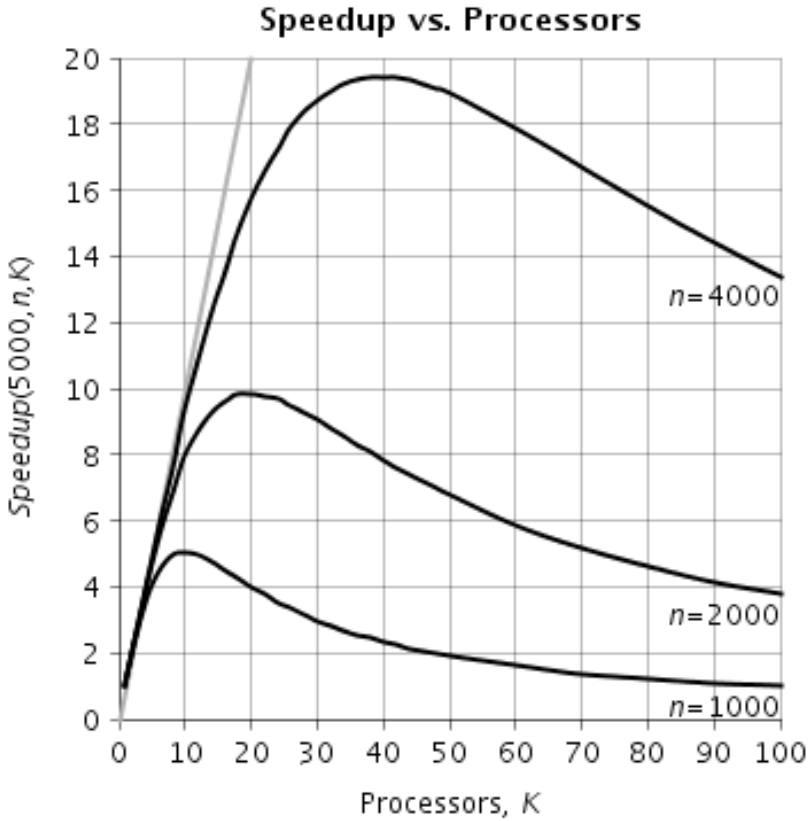
- calculation time decreases as a function of K
- communication time increases as a function of K
  - but now $O(K)$, rather than only $O(\log_2 K)$

Running Time Model



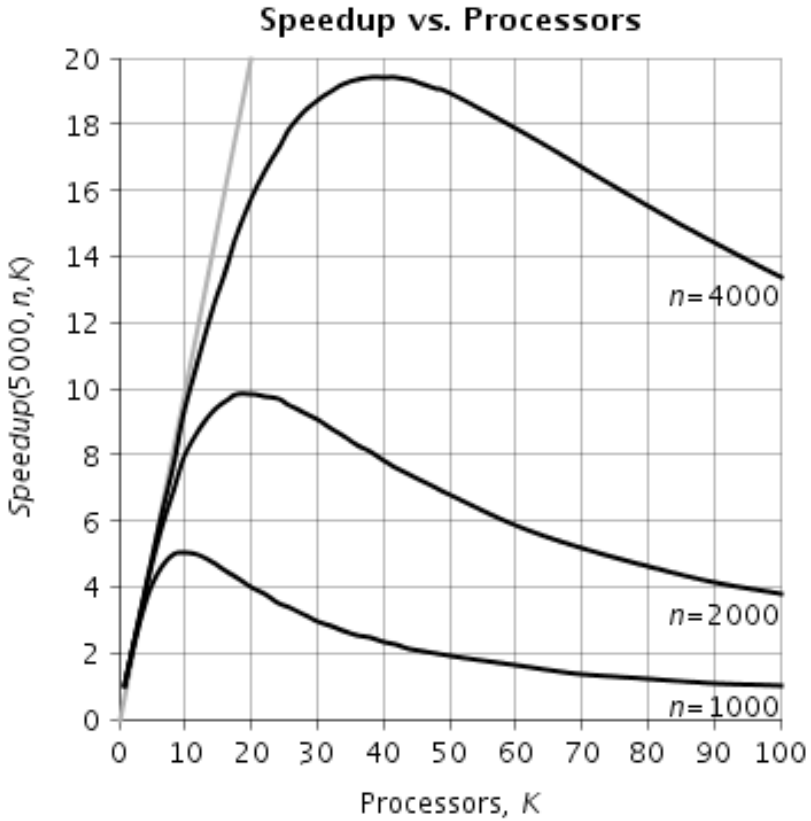
$$T^{\bar{p}}(s,n,K) = 2.04\times10^{-8}sn^2\frac{1}{K} + s(2.08\times10^{-4} + 1.37\times10^{-7}n\frac{1}{K})(K-1)$$

Speedup Model



$$Speedup^{\bar{p}}(s, n, K) = \frac{T^{\bar{p}}(s, n, 1)}{T^{\bar{p}}(s, n, K)}$$

Speedup Model



What is the maximum speedup that the program can acheive?

Speedup Model

What is the maximum speedup that the program can acheive?

Speedup Model

What is the maximum speedup that the program can acheive?

Find the value of K that results in the minimum running time.

Speedup Model

What is the maximum speedup that the program can acheive?

Find the value of K that results in the minimum running time.

Differentiate $T^{\bar{P}}(s, n, K)$ with respect to K,
set the derivative equal to 0, and solve for K.

Speedup Model

$$K^{\bar{p}}_{\text{best}}(s,n) = \sqrt{(an^2 - \tfrac{1}{B}128n)/L}$$

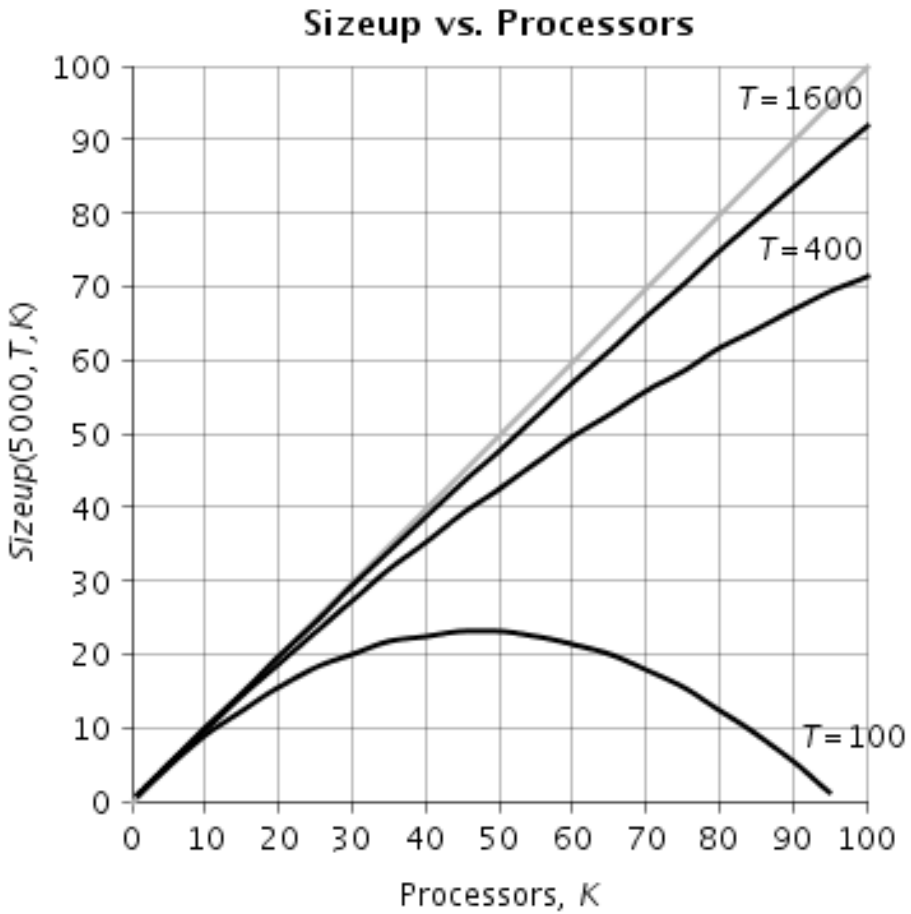
$$K^{\bar{p}}_{\text{best}}(s,n) = \sqrt{9.81\times10^{-5}n^2 - 6.59\times10^{-4}n}$$

$$K^{\bar{p}}_{\text{best}}(5000,2000) \approx 20$$

$$Speedup^{\bar{p}}(5000,2000,20) = 9.891$$

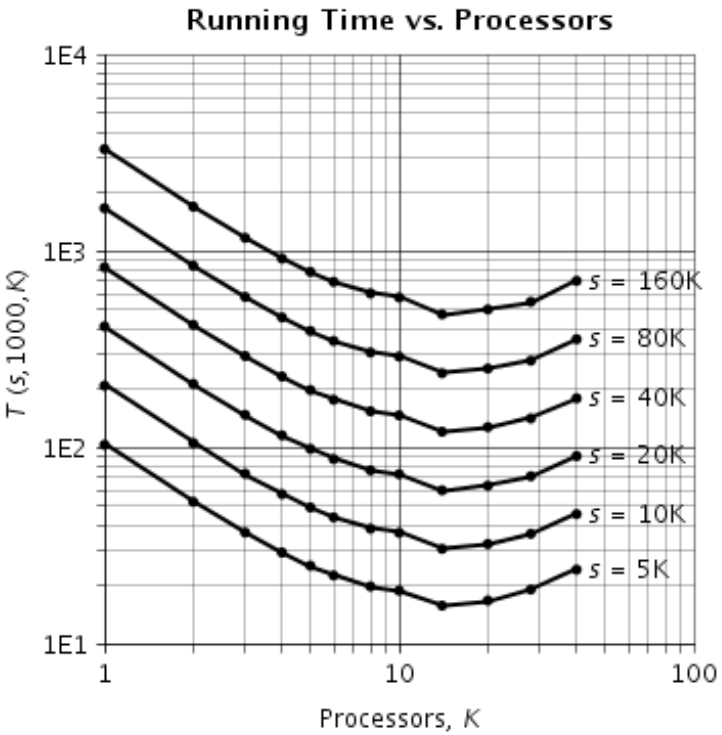
$$Eff^{\bar{p}}(5000.2000,20) = 0.495$$

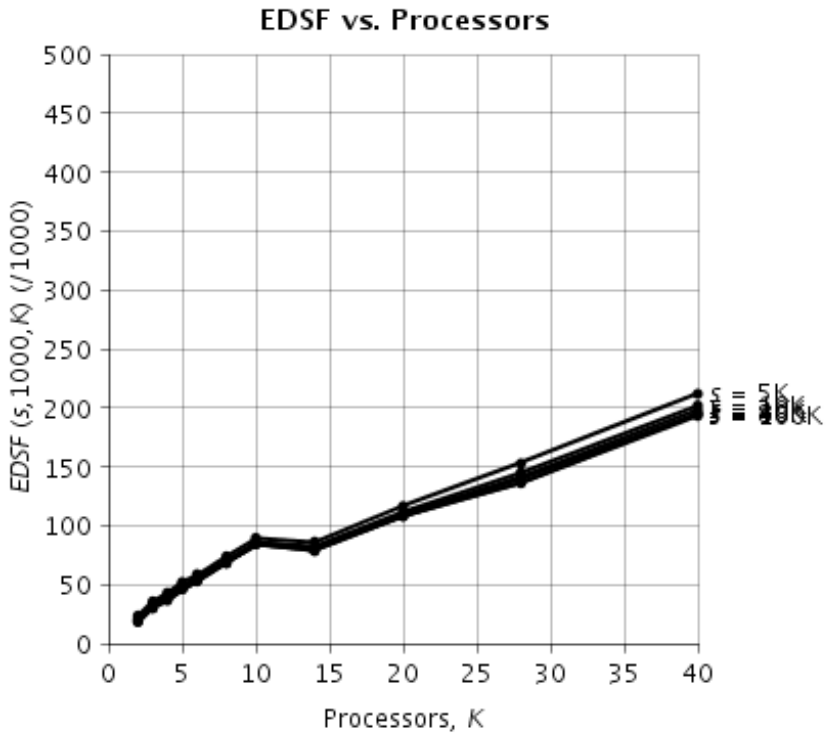
Sizeup Model



$$Sizeup^{\bar{p}}(s, T, K) = \frac{N^{\bar{p}}(s, T, K)}{N^{\bar{p}}(s, T, 1)} = \left(\frac{n^{\bar{p}}(s, T, K)}{n^{\bar{p}}(s, T, 1)}\right)^2$$

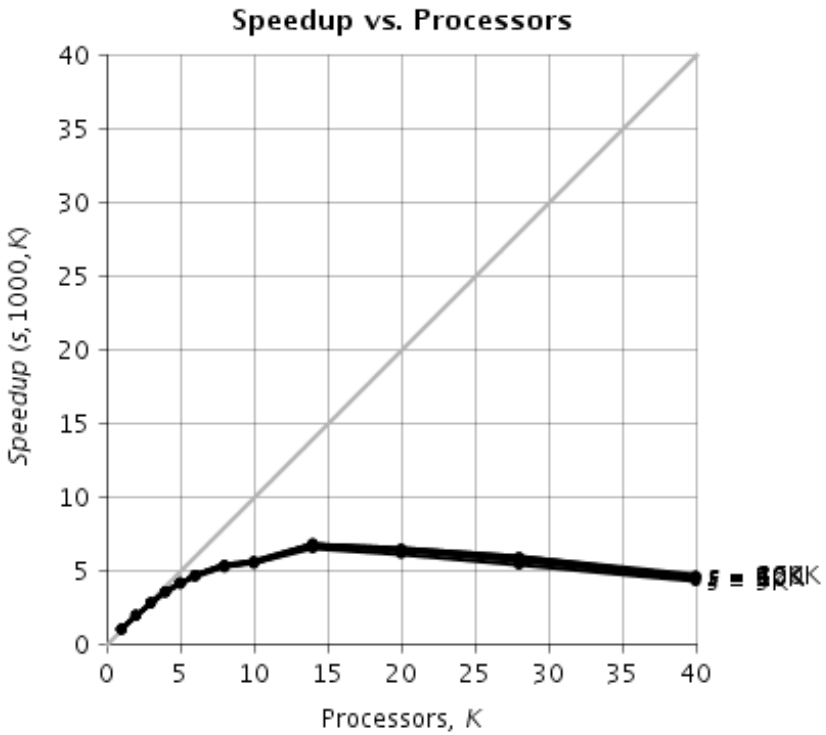
`AntiprotonClu` Running Time and EDSF

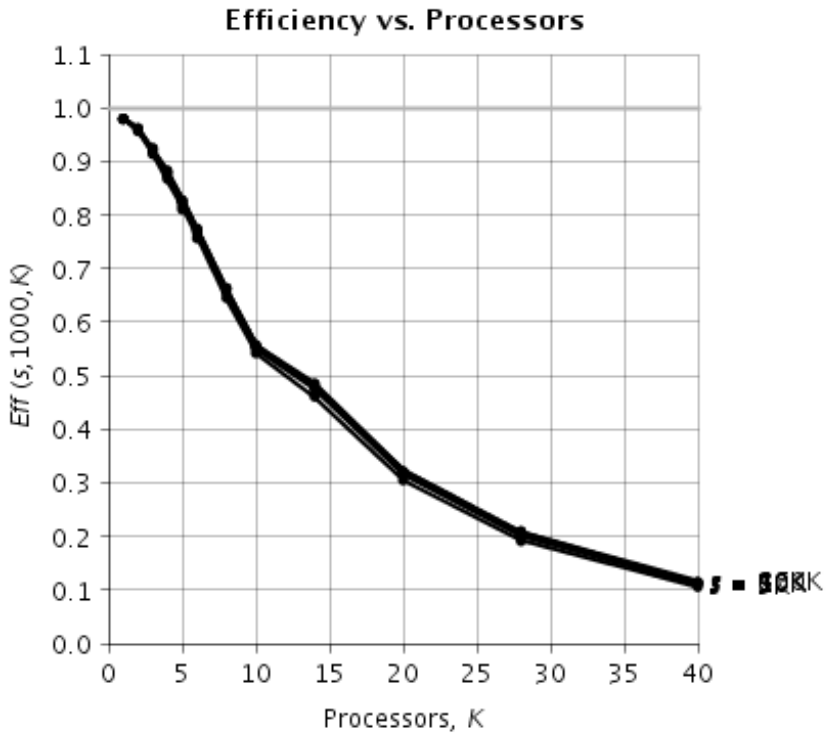




s varying; $n = 1000$

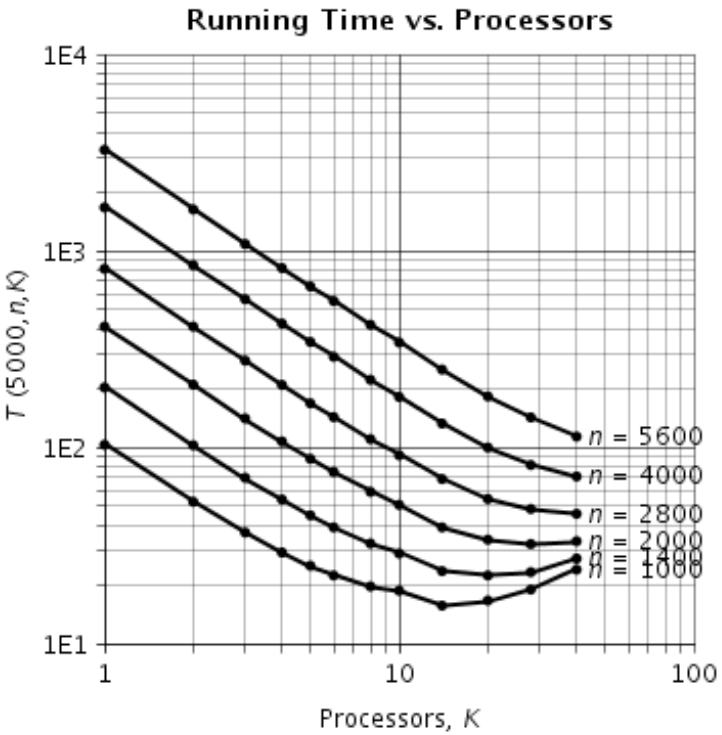
AntiprotonClu Speedup and Efficiency

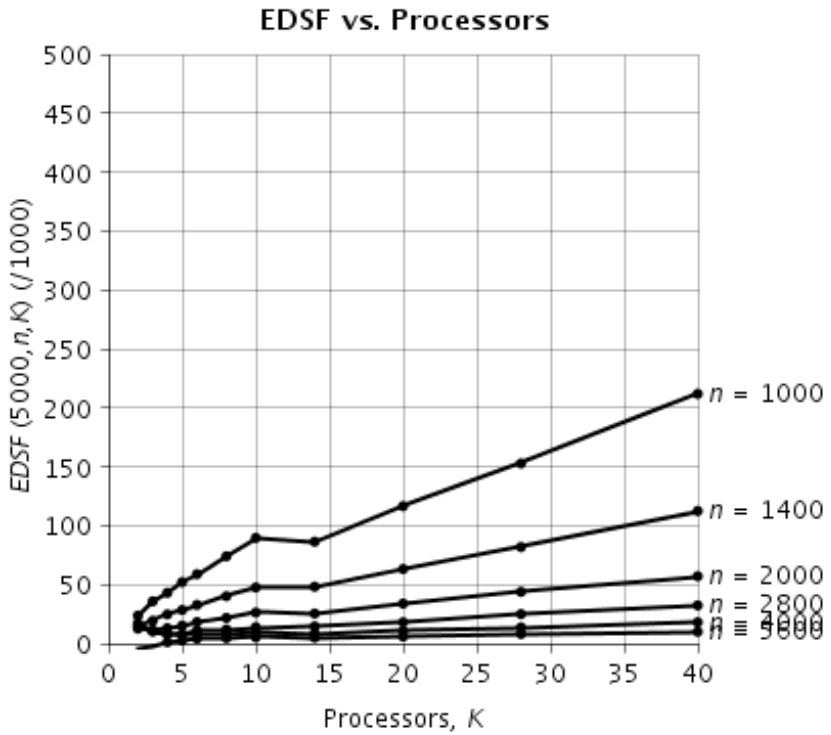




s varying; $n = 1000$

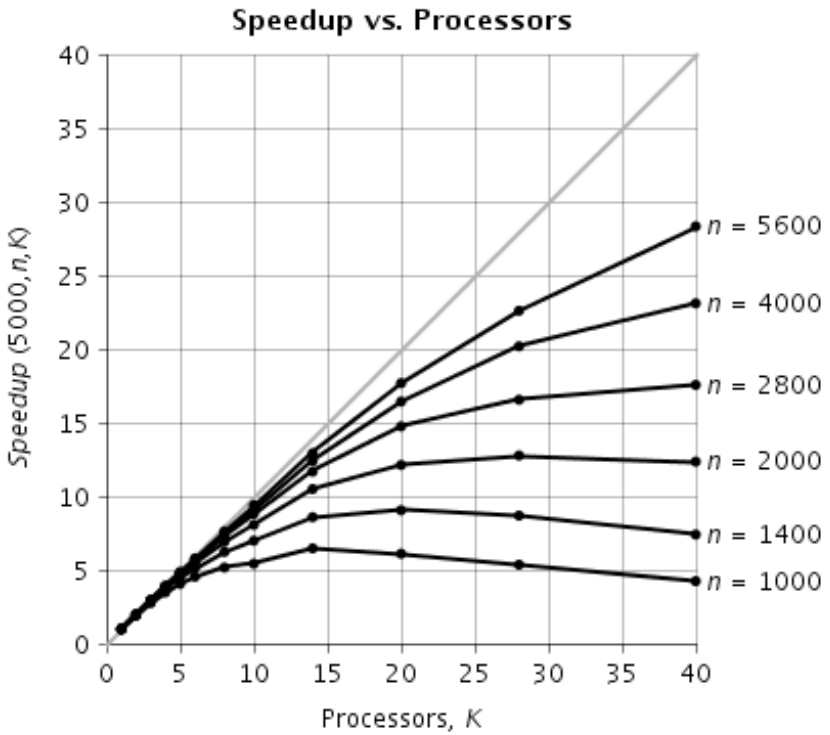
AntiprotonClu Running Time and EDSF

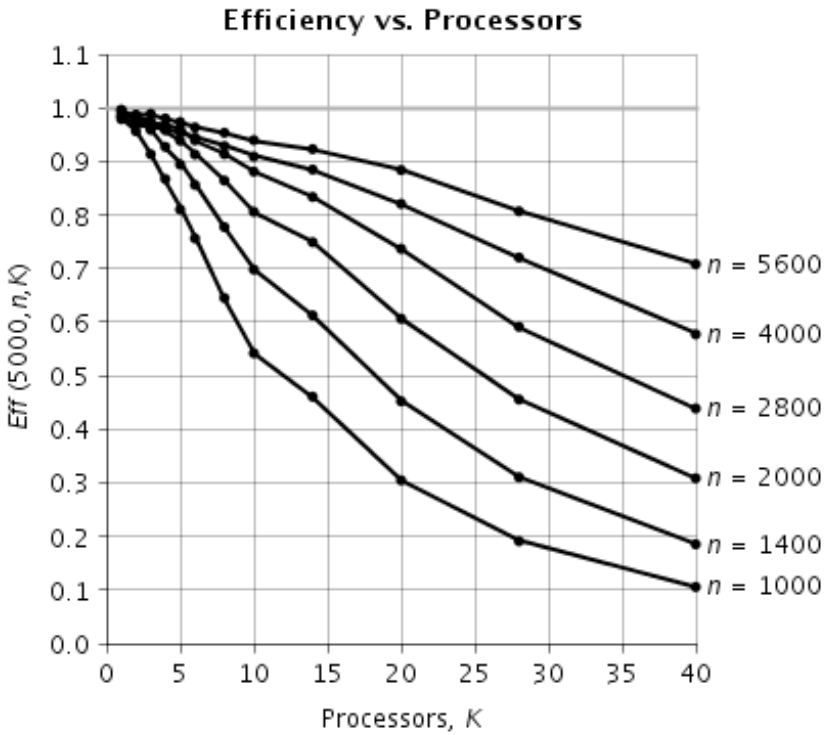




$s = 5000$; n varying

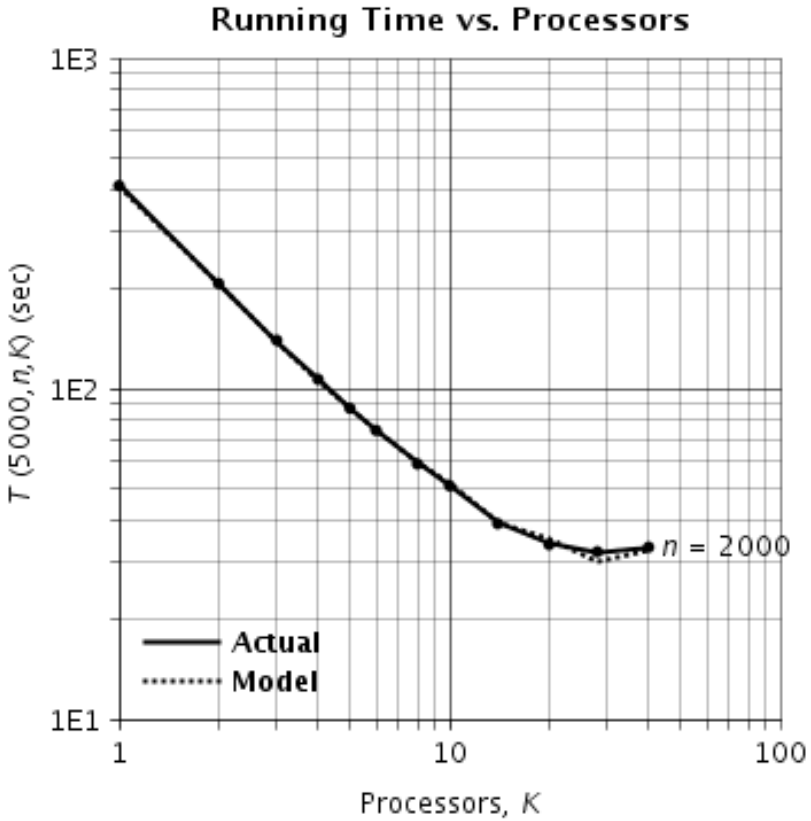
AntiprotonClu Speedup and Efficiency





$s = 5000$; n varying

AntiprotonClu Experimental Results



$s = 5000$; $n = 2000$

Scalability

Parallel computing:

- bigger problems
- faster answers

Scalability

Parallel computing:

- bigger problems
- faster answers

Scalability: the *ability* of a parallel program to calculate larger problems as the number of processors *scales up* (increases).

- a *qualitative* quantity
- influenced by a number of *quantitative* quantities

Scalability

Scalability: the *ability* of a parallel program to calculate larger problems as the number of processors *scales up* (increases).

- isoefficiency function: formula for N as a function of K such that the parallel program's efficiency is held the same.
- size function ($N(T, K)$): formula for N as a function of K such that the parallel program's running time is held the same.
  - problem size N is proportional to amount of computation (by defn.)

Scalability

Scalability: the *ability* of a parallel program to calculate larger problems as the number of processors *scales up* (increases).

- input-size function ($n(T, K)$): formula for n as a function K such that the parallel program's running time is held the same.

Scalability

Scalability: the *ability* of a parallel program to calculate larger problems as the number of processors *scales up* (increases).

- input-size function ($n(T, K)$): formula for n as a function K such that the parallel program's running time is held the same.
  - problem size N is a function f of input size n
    - AES key search: $f(n) = 2^n$
    - Floyd's algorithm: $f(n) = n^3$
    - Antiproton motion: $f(n) = n^2$

Scalability

Scalability: the *ability* of a parallel program to calculate larger problems as the number of processors *scales up* (increases).

- input-size function ($n(T, K)$): formula for n as a function K such that the parallel program's running time is held the same.
  - problem size N is a function f of input size n
    - AES key search: $f(n) = 2^n$
    - Floyd's algorithm: $f(n) = n^3$
    - Antiproton motion: $f(n) = n^2$
  - input size n is a function f^{-1} of problem size N
    - AES key search: $f^{-1}(N) = \log_2 N$
    - Floyd's algorithm: $f^{-1}(N) = \sqrt[3]{N}$
    - Antiproton motion: $f^{-1}(N) = \sqrt{N}$

Scalability

Scalability: the *ability* of a parallel program to calculate larger problems as the number of processors *scales up* (increases).

- input-size function ($n(T, K)$): formula for n as a function K such that the parallel program's running time is held the same.
  - problem size N is a function f of input size n
    - AES key search: $f(n) = 2^n$
    - Floyd's algorithm: $f(n) = n^3$
    - Antiproton motion: $f(n) = n^2$
  - input size n is a function f^{-1} of problem size N
    - AES key search: $f^{-1}(N) = \log_2 N$
    - Floyd's algorithm: $f^{-1}(N) = \sqrt[3]{N}$
    - Antiproton motion: $f^{-1}(N) = \sqrt{N}$
  - $n(T, K) = f^{-1}(N(T, K))$

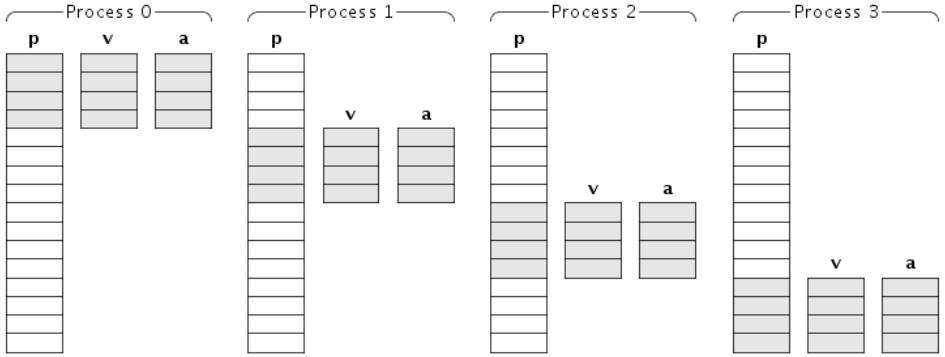
Scalability

Scalability: the *ability* of a parallel program to calculate larger problems as the number of processors *scales up* (increases).

To solve a given problem size, program requires both *time* and *space*. Often, scalability is limited by the memory required by each process.

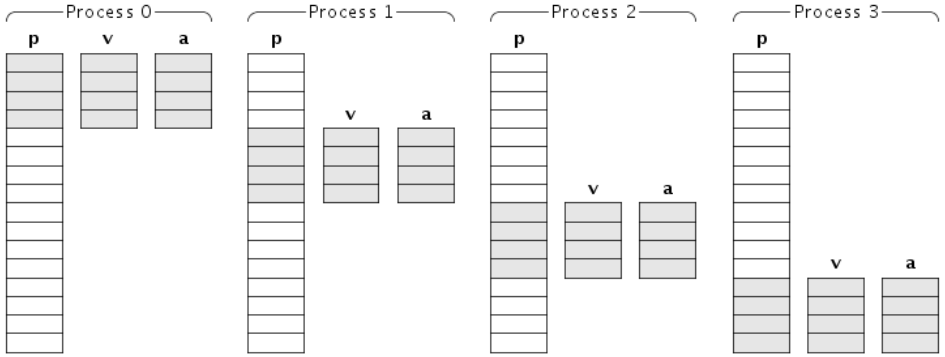
Scalability

Often, scalability is limited by the memory required by each process.



Scalability

Often, scalability is limited by the memory required by each process.

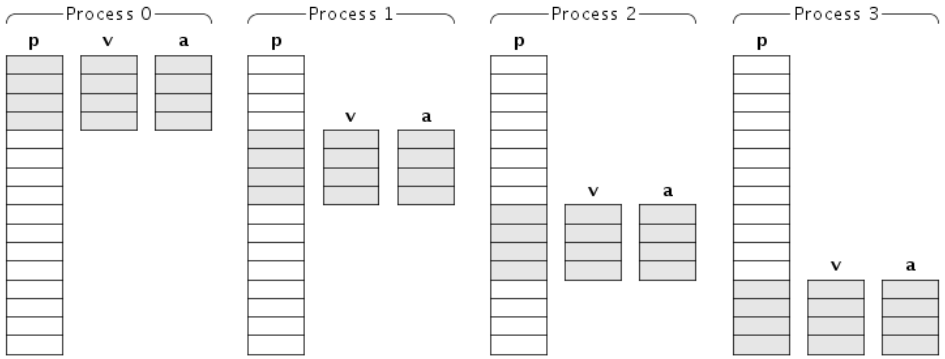


Memory (in each process) for

- acceleration array is $O(n/K)$
- velocity array is $O(n/K)$
- position array is $O(n)$

Scalability

Often, scalability is limited by the memory required by each process.



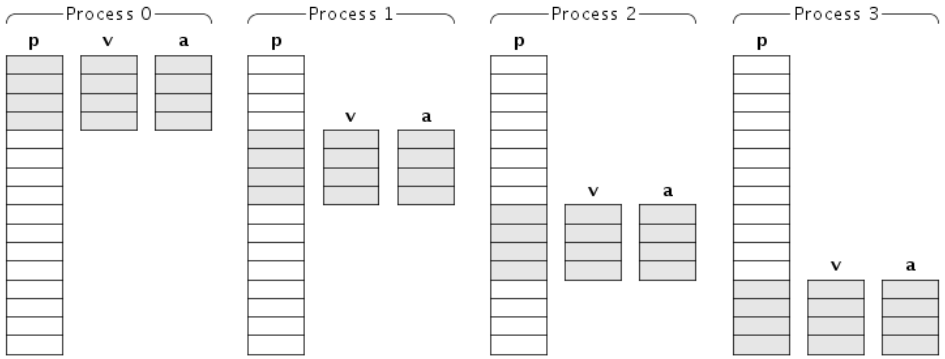
Memory (in each process) for

- acceleration array is $O(n/K) = O(\sqrt{K}/K) = O(1/\sqrt{K})$
- velocity array is $O(n/K) = O(\sqrt{K}/K) = O(1/\sqrt{K})$
- position array is $O(n) = O(\sqrt{K})$

For a fixed running time T and assuming ideal sizeup, n is $O(\sqrt{K})$.

Scalability

Often, scalability is limited by the memory required by each process.



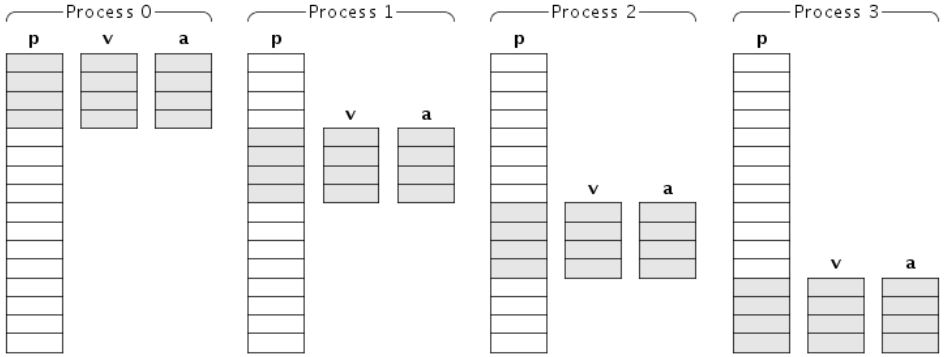
Memory (in each process) for

- acceleration array is $O(n/K) = O(\sqrt{K}/K) = O(1/\sqrt{K})$
- velocity array is $O(n/K) = O(\sqrt{K}/K) = O(1/\sqrt{K})$
- position array is $O(n) = O(\sqrt{K})$

For a fixed running time T and assuming ideal sizeup, n is $O(\sqrt{K})$.

Scalability

Often, scalability is limited by the memory required by each process.



Memory (in each process) for

- acceleration array is $O(n/K) = O(\sqrt{K}/K) = O(1/\sqrt{K})$
- velocity array is $O(n/K) = O(\sqrt{K}/K) = O(1/\sqrt{K})$
- position array is $O(n) = O(\sqrt{K})$

For a fixed running time T and assuming ideal sizeup, n is $O(\sqrt{K})$.
In the limit, as $K \to \infty$?

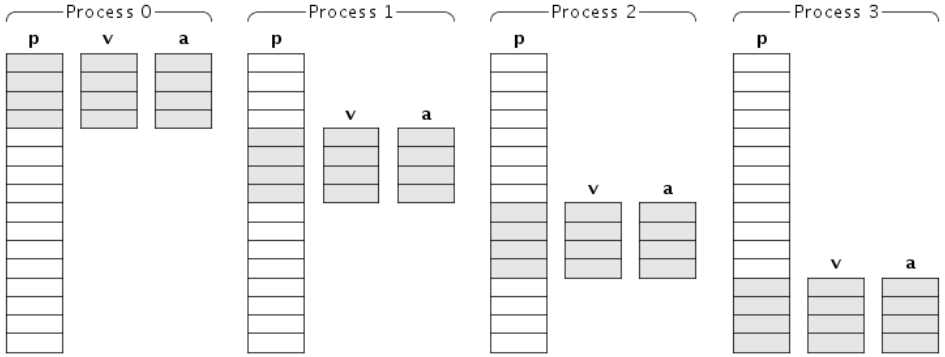
Scalability

Often, scalability is limited by the memory required by each process.

If $K_{\mathbf{best}}$ (for either speedup or sizeup)
implies memory required by each process exceeds available memory,
then can't acheive best performance.

Scalability

Often, scalability is limited by the memory required by each process.

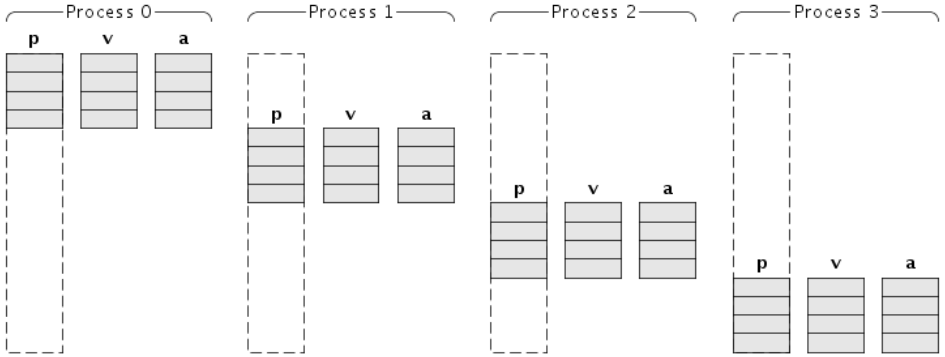


Memory (in each process) for

- acceleration array is $O(n/K) = O(\sqrt{K}/K) = O(1/\sqrt{K})$
- velocity array is $O(n/K) = O(\sqrt{K}/K) = O(1/\sqrt{K})$
- position array is $O(n) = O(\sqrt{K}) = O(\sqrt{K})$

Scalability

Often, scalability is limited by the memory required by each process.

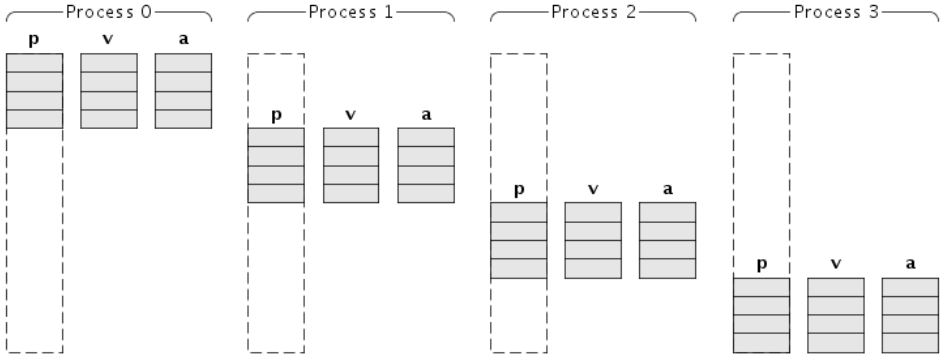


Memory (in each process) for

- acceleration array is $O(n/K) = O(\sqrt{K}/K) = O(1/\sqrt{K})$
- velocity array is $O(n/K) = O(\sqrt{K}/K) = O(1/\sqrt{K})$
- position array is $O(n/K) = O(\sqrt{K}/K) = O(1/\sqrt{K})$

Scalability

Often, scalability is limited by the memory required by each process.



Memory (in each process) for

- acceleration array is $O(n/K) = O(\sqrt{K}/K) = O(1/\sqrt{K})$
- velocity array is $O(n/K) = O(\sqrt{K}/K) = O(1/\sqrt{K})$
- position array is $O(n/K) = O(\sqrt{K}/K) = O(1/\sqrt{K})$

But, every process needs *all* positions to compute electrostatic forces.

Pipelined Message Passing

Rearrange the program's calculation and communication.

Current arrangement:

- Calculate *all* of the electrostatic forces, then
- Commmunicate *all* of the particle positions (with `allGather`)

Pipelined Message Passing

Rearrange the program's calculation and communication.

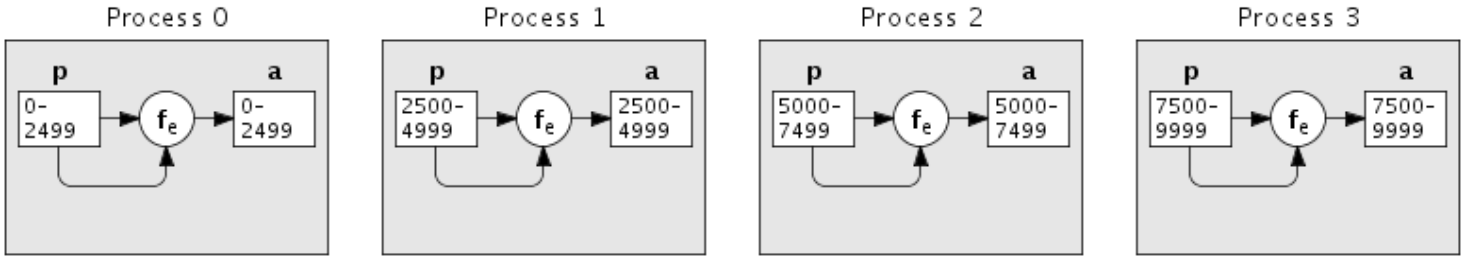
Current arrangement:

- Calculate *all* of the electrostatic forces, then
- Commmunicate *all* of the particle positions (with `allGather`)

Desired arragement:

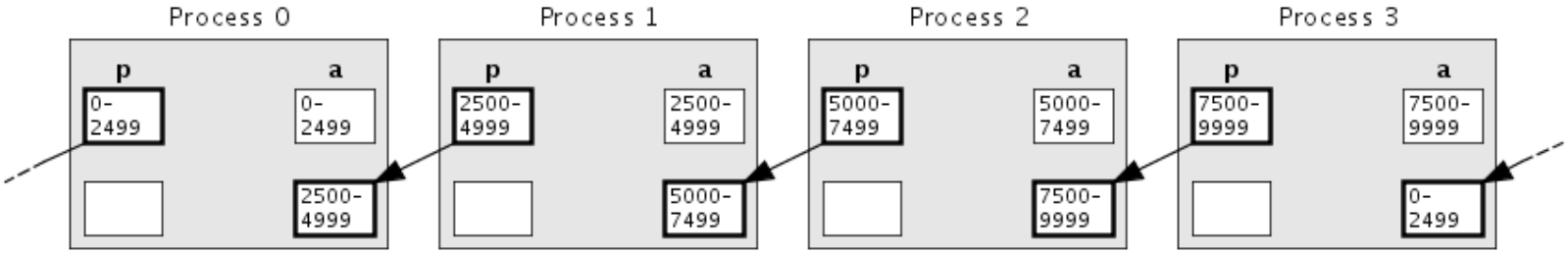
- Calculate *a slice* of the electrostatic forces, then
- Communicate *a slice* of the particle positions, then
- Loop

Pipelined Message Passing



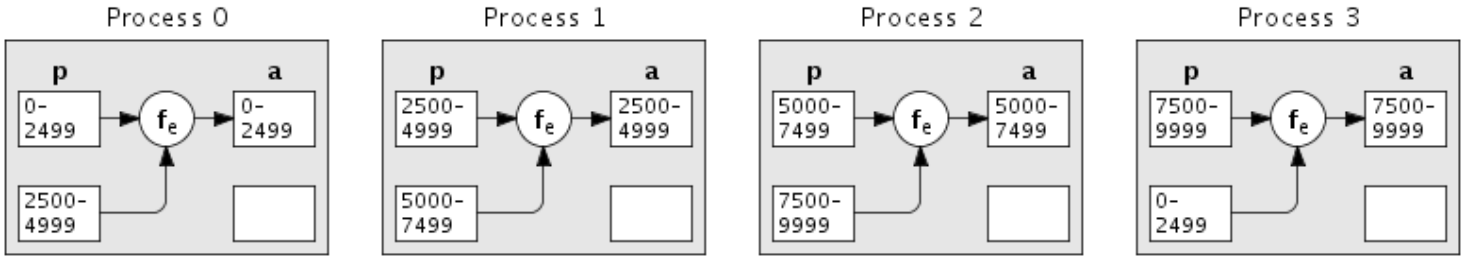
Compute the electrostatic forces between all particles in this process's slice and all particles in this process's slice,
acculumating the forces into the acceleration slice.

Pipelined Message Passing



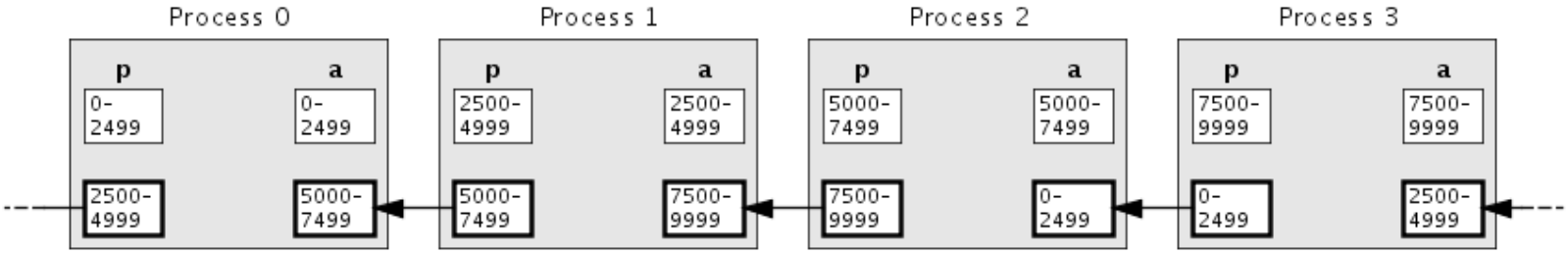
Send this process's slice of positions to the previous process and receive the next process's slice of positions into auxiliary buffer. (There is also another auxiliary buffer.)

Pipelined Message Passing



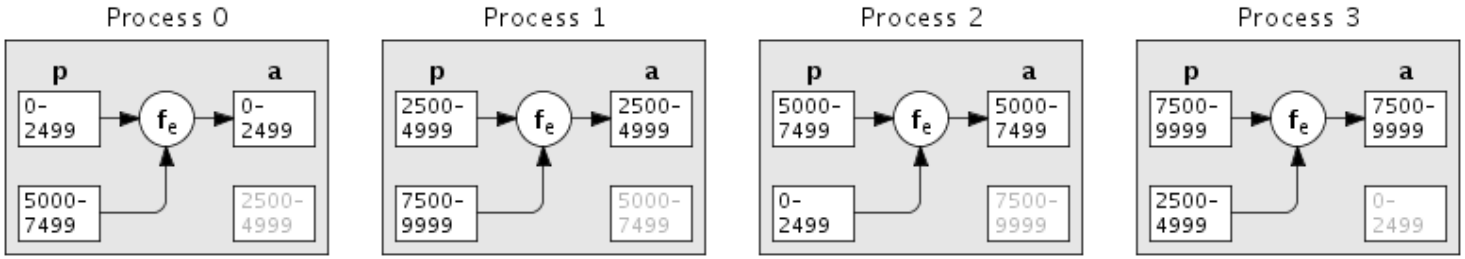
Swap the auxiliary buffers.
Compute the electrostatic forces between all particles in this process's slice and all particles in the just-received slice of positions,
acculumating the forces into the acceleration slice.

Pipelined Message Passing



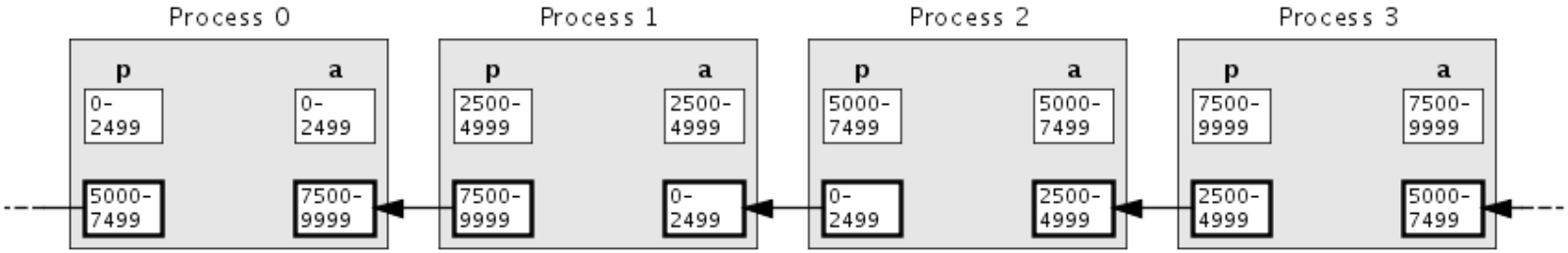
Send the slice of positions in 1^{st} auxiliary buffer to the previous process and receive a slice of positions from the next process into 2^{nd} auxiliary buffer.

Pipelined Message Passing



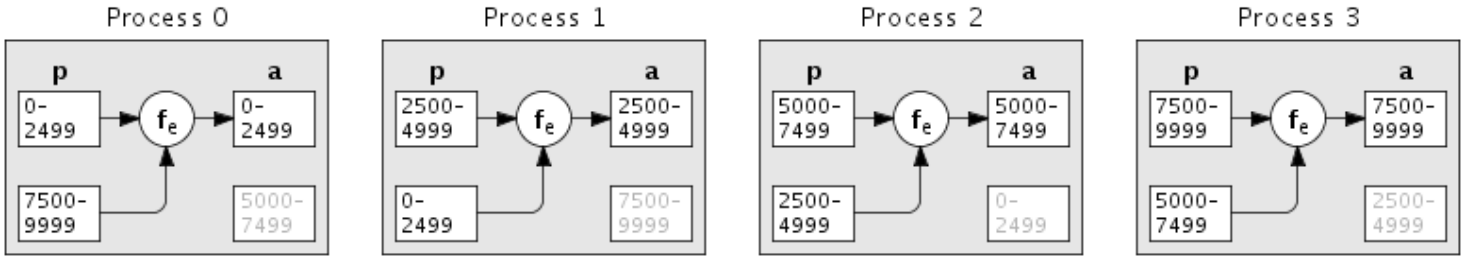
Swap the auxiliary buffers.
Compute the electrostatic forces between all particles in this process's slice and all particles in the just-received slice of positions,
acculumating the forces into the acceleration slice.

Pipelined Message Passing



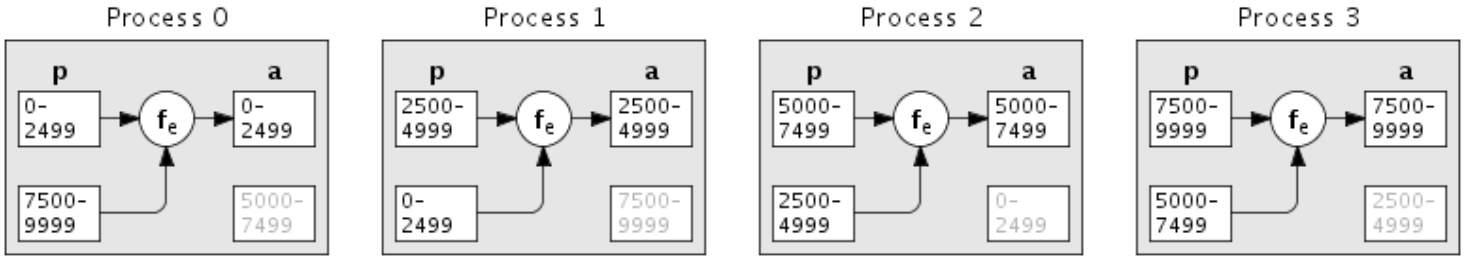
Send the slice of positions in 1^{st} auxiliary buffer to the previous process and receive a slice of positions from the next process into 2^{nd} auxiliary buffer.

Pipelined Message Passing



Swap the auxiliary buffers.
Compute the electrostatic forces between all particles in this process's slice and all particles in the just-received slice of positions,
acculumating the forces into the acceleration slice.

Pipelined Message Passing

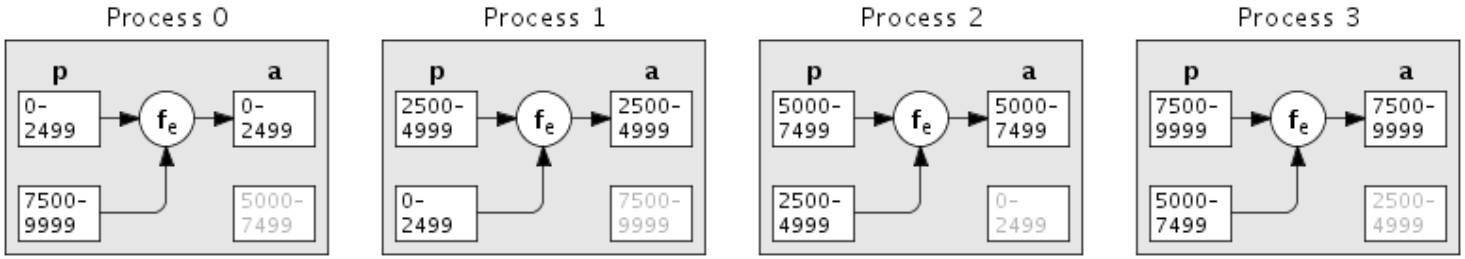


Swap the auxiliary buffers.
Compute the electrostatic forces between all particles in this process's slice and all particles in the just-received slice of positions,
acculumating the forces into the acceleration slice.

Every process must send a slice of positions to one process
and receive a slice of positions from another process.

Which collective communication operation?

Pipelined Message Passing



Swap the auxiliary buffers.
Compute the electrostatic forces between all particles in this process's slice and all particles in the just-received slice of positions, acculumating the forces into the acceleration slice.

Every process must send a slice of positions to one process and receive a slice of positions from another process.

Which collective communication operation? send-receive

Send-Receive Implementation and Message Time Model

```
int toRank = (rank - 1 + size) % size;
int fromRank = (rank + 1) % size;
world.sendReceive (toRank, outBuf, fromRank, inBuf);
```

Send-Receive Implementation and Message Time Model

```
int toRank = (rank - 1 + size) % size;
int fromRank = (rank + 1) % size;
world.sendReceive (toRank, outBuf, fromRank, inBuf);
```

Sends the outgoing message and receives the incoming message simultaneousy in separate threads.

Send-Receive Implementation and Message Time Model

```
int toRank = (rank - 1 + size) % size;
int fromRank = (rank + 1) % size;
world.sendReceive (toRank, outBuf, fromRank, inBuf);
```

Sends the outgoing message and receives the incoming message simultaneousy in separate threads.

$$T_{\textbf{send-receive}}(b, K) =$$

Send-Receive Implementation and Message Time Model

```
int toRank = (rank - 1 + size) % size;
int fromRank = (rank + 1) % size;
world.sendReceive (toRank, outBuf, fromRank, inBuf);
```

Sends the outgoing message and receives the incoming message simultaneousy in separate threads.

$$T_{\textbf{send-receive}}(b, K) = (L + \frac{1}{B}b)$$

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

AntiprotonClu2.java

code/AntiprotonClu2.java

AntiprotonClu2.java

code/AntiprotonClu2.java

Before looking at the running-time measurements for AntiprotonClu2, derive a model to predict the running time.

What will we need?

- a calculation time model (**same as** AntiprotonClu)
- a communication time model

Calculation Time Model

Antiproton motion algorithm is $O(sn^2)$,
where n is the number of particles
and s is total number of steps ($s = snaps \cdot steps$).

$$T^{\bar{p}2}_{\mathbf{calc}}(s, n, 1) = asn^2$$

$$T^{\bar{p}2}_{\mathbf{calc}}(s, n, K) = asn^2 \frac{1}{K}$$

$$T^{\bar{p}2}_{\mathbf{calc}}(s, n, 1) = 2.04 \times 10^{-8} sn^2$$

$$T^{\bar{p}2}_{\mathbf{calc}}(s, n, K) = 2.04 \times 10^{-8} sn^2 \frac{1}{K}$$

Communication Time Model

- How many send-receives are performed?
- How many bits in each send-receives message?

Communication Time Model

- How many send-receives are performed? $s(K-1)$
- How many bits in each send-receives message? $128n\frac{1}{K}$

Communication Time Model

- How many send-receives are performed? $s(K-1)$
- How many bits in each send-receives message? $128n\frac{1}{K}$

$$T^{\bar{p}2}_{\text{comm}}(s,n,K) = s(K-1)\,T_{\text{send-receive}}(128n\tfrac{1}{K},K)$$

$$T^{\bar{p}2}_{\text{comm}}(s,n,K) = s(K-1)(L+\tfrac{1}{B}128n\tfrac{1}{K})$$

$$T^{\bar{p}2}_{\text{comm}}(s,n,K) = s(K-1)(2.08\times10^{-4}+1.37\times10^{-7}n\tfrac{1}{K})$$

Communication Time Model

- How many send-receives are performed? $s(K-1)$
- How many bits in each send-receives message? $128n\frac{1}{K}$

$$T^{\bar{p}2}_{\text{comm}}(s,n,K) = s(K-1)T_{\text{send-receive}}(128n\tfrac{1}{K},K)$$

$$T^{\bar{p}2}_{\text{comm}}(s,n,K) = s(K-1)(L+\tfrac{1}{B}128n\tfrac{1}{K})$$

$$T^{\bar{p}2}_{\text{comm}}(s,n,K) = s(K-1)(2.08\times10^{-4}+1.37\times10^{-7}n\tfrac{1}{K})$$

Note: $T^{\bar{p}2}_{\text{comm}}(s,n,K) = T^{\bar{p}}_{\text{comm}}(s,n,K)$.

Communication Time Model

- How many send-receives are performed? $s(K-1)$
- How many bits in each send-receives message? $128n\frac{1}{K}$

$$T^{\bar{p}2}_{\text{comm}}(s,n,K) = s(K-1)\,T_{\text{send-receive}}(128n\tfrac{1}{K},K)$$

$$T^{\bar{p}2}_{\text{comm}}(s,n,K) = s(K-1)(L+\tfrac{1}{B}128n\tfrac{1}{K})$$

$$T^{\bar{p}2}_{\text{comm}}(s,n,K) = s(K-1)(2.08\times10^{-4}+1.37\times10^{-7}n\tfrac{1}{K})$$

Note: $T^{\bar{p}2}_{\text{comm}}(s,n,K) = T^{\bar{p}}_{\text{comm}}(s,n,K)$.

`AntiprotonClu2`'s send-receive messages are
exactly the same as `AntiprotonClu`'s all-gather messages;
the former does them interpsersed with calculation,
while the later does them all at once.

Running Time Model

$$T^{\bar{p}2}(s,n,K) = T^{\bar{p}2}_{\text{calc}}(s,n,K) + T^{\bar{p}2}_{\text{comm}}(s,n,K)$$

$$T^{\bar{p}2}(s,n,K) = asn^2\frac{1}{K} + s(L + \frac{1}{B}128n\frac{1}{K})(K-1)$$

$$T^{\bar{p}2}(s,n,K) = 2.04\times10^{-8}sn^2\frac{1}{K} + s(K-1)(2.08\times10^{-4} + 1.37\times10^{-7}n\frac{1}{K})$$

Running Time Model

$$T^{\bar{p}2}(s,n,K) = T^{\bar{p}2}_{\text{calc}}(s,n,K) + T^{\bar{p}2}_{\text{comm}}(s,n,K)$$

$$T^{\bar{p}2}(s,n,K) = asn^2\frac{1}{K} + s(L + \frac{1}{B}128n\frac{1}{K})(K-1)$$

$$T^{\bar{p}2}(s,n,K) = 2.04\times10^{-8}sn^2\frac{1}{K} + s(K-1)(2.08\times10^{-4} + 1.37\times10^{-7}n\frac{1}{K})$$

Note: $T^{\bar{p}2}(s,n,K) = T^{\bar{p}}(s,n,K)$.

Predict that `AntiprotonClu2` has
exactly the same performance as `AntiprotonClu`.

Running Time Model

$$T^{\bar{p}2}(s,n,K) = T^{\bar{p}2}_{\text{calc}}(s,n,K) + T^{\bar{p}2}_{\text{comm}}(s,n,K)$$

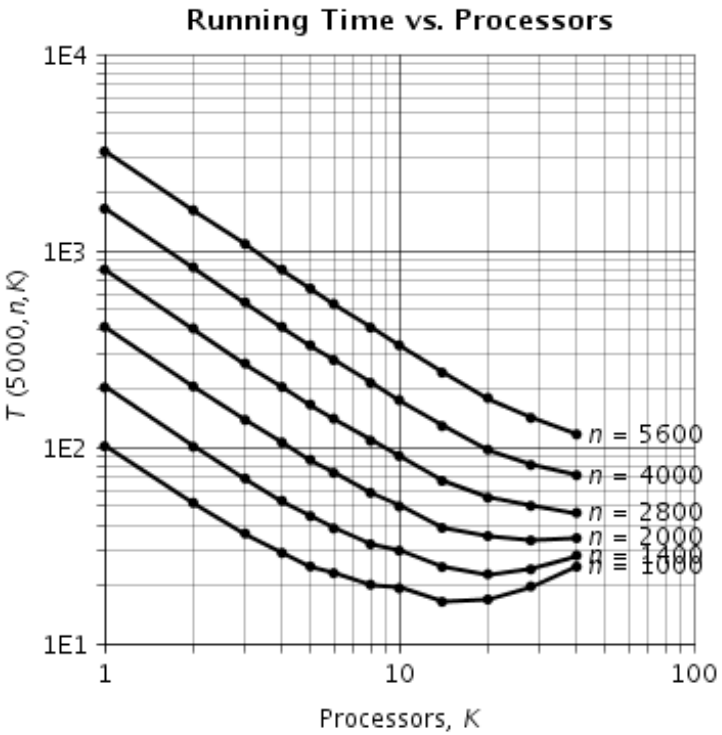
$$T^{\bar{p}2}(s,n,K) = asn^2\frac{1}{K} + s(L + \frac{1}{B}128n\frac{1}{K})(K-1)$$

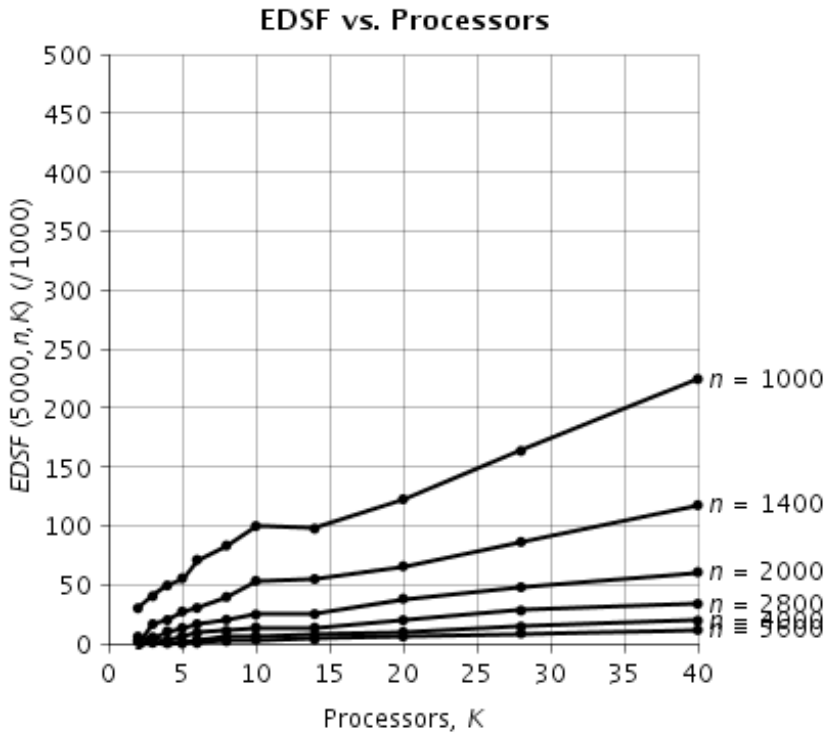
$$T^{\bar{p}2}(s,n,K) = 2.04\times10^{-8}sn^2\frac{1}{K} + s(K-1)(2.08\times10^{-4} + 1.37\times10^{-7}n\frac{1}{K})$$

Note: $T^{\bar{p}2}(s,n,K) = T^{\bar{p}}(s,n,K)$.

Predict that `AntiprotonClu2` has
exactly the same performance as `AntiprotonClu`.
But former has significantly improved memory scalability.

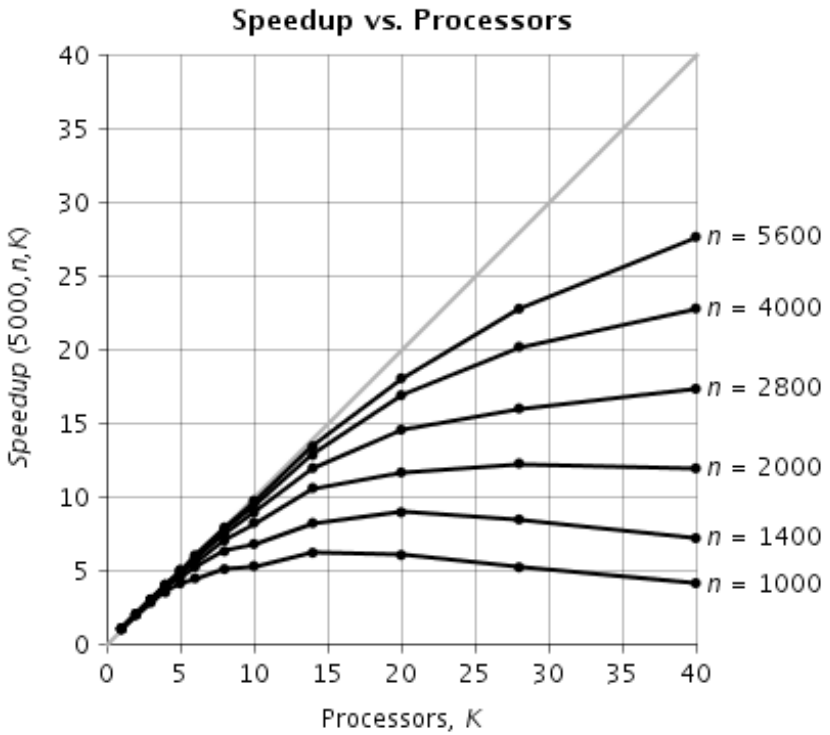
AntiprotonClu2 Running Time and EDSF

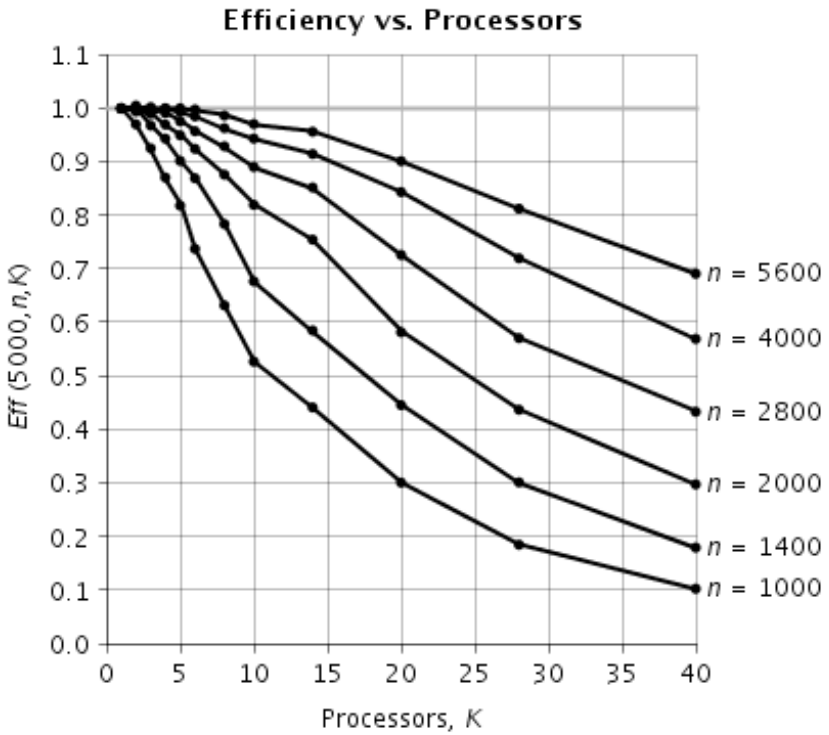




$s = 5000$; n varying

AntiprotonClu2 Speedup and Efficiency

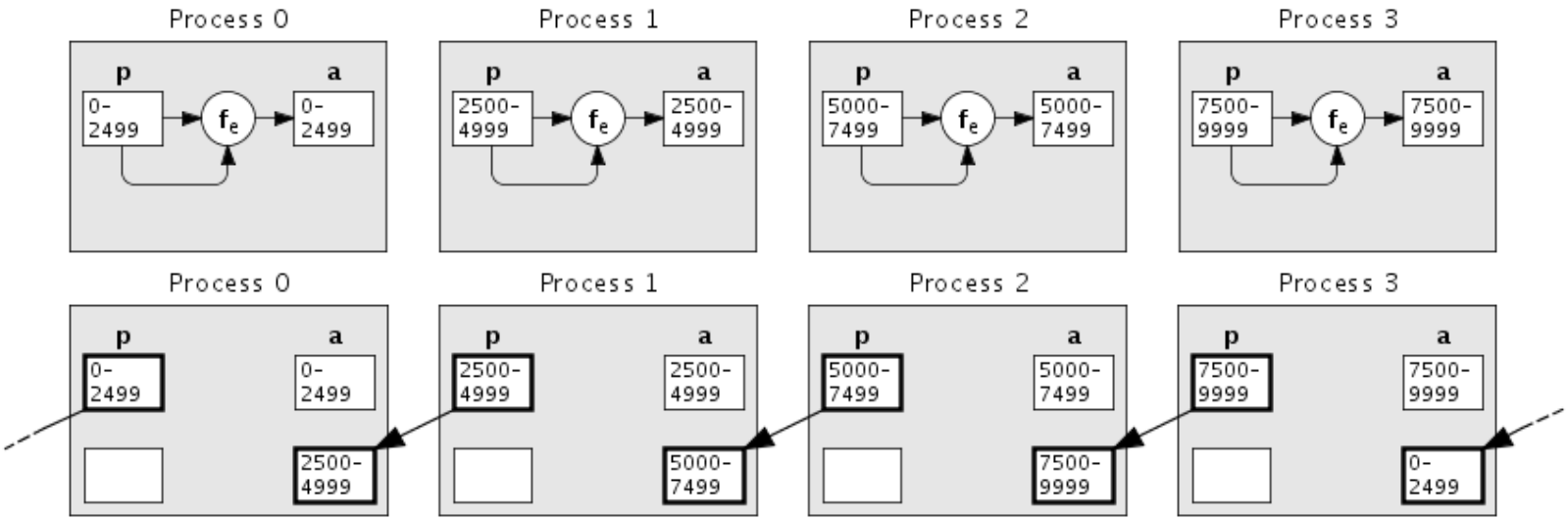




$s = 5000$; n varying

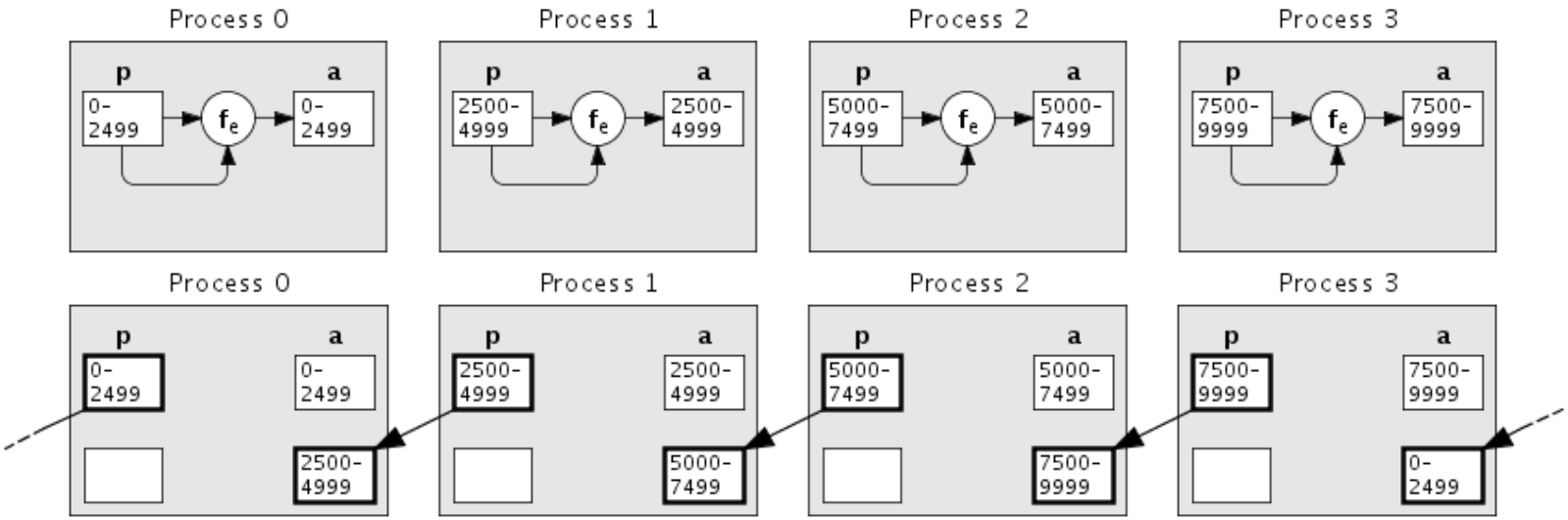
Interspersed Computation and Communication

`AntiprotonClu2` improved the memory scalability
through pipelined message passing,
which interpsersed computation and communication.



Interspersed Computation and Communication

`AntiprotonClu2` improved the memory scalability
through pipelined message passing,
which interpsersed computation and communication.



While communicating, process is not computing!

Overlapped Computation and Communication

Further rearrange the program's calculation and communication.

Current arrangement:

- Calculate a slice of the electrostatic forces, *then*
- Communicate a slice of the particle positions, then
- Loop

Overlapped Computation and Communication

Further rearrange the program's calculation and communication.

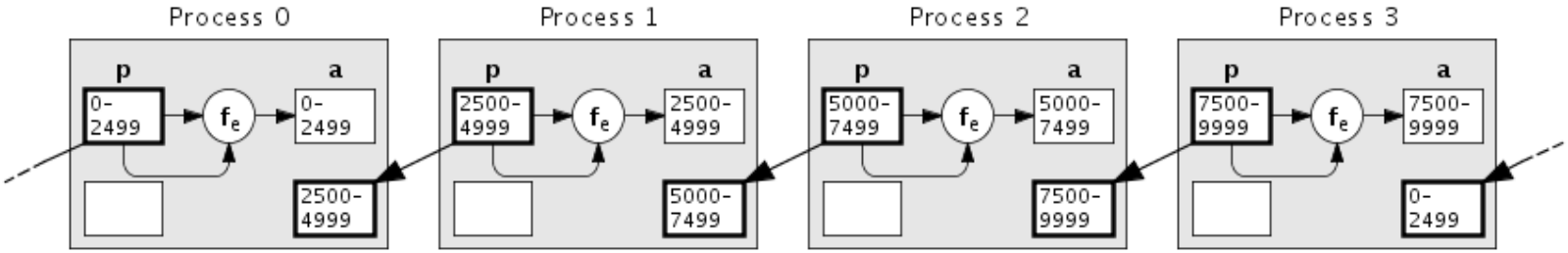
Current arrangement:

- Calculate a slice of the electrostatic forces, *then*
- Communicate a slice of the particle positions, then
- Loop

Desired arragement:

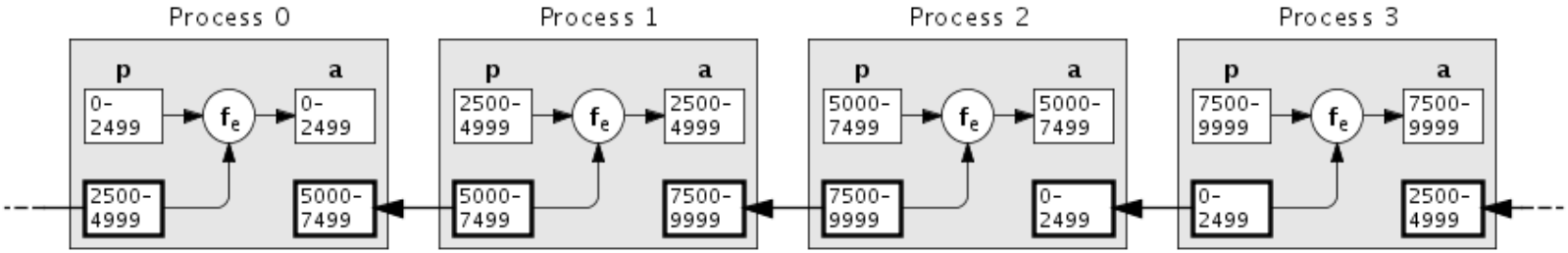
- Calculate a slice of the electrostatic forces, *and*
- Communicate a slice of the particle positions, then
- Loop

Overlapped Computation and Communication



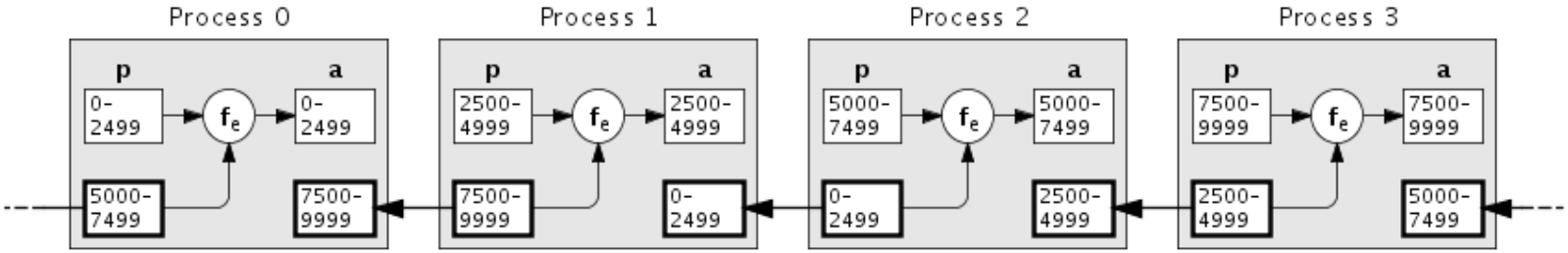
Compute the electrostatic forces between all particles in this process's slice and all particles in this process's slice,
acculumating the forces into the acceleration slice;
and, at the same time,
send this process's slice of positions to the previous process and receive the next process's slice of positions into auxiliary buffer.
(There is also another auxiliary buffer.)

Overlapped Computation and Communication



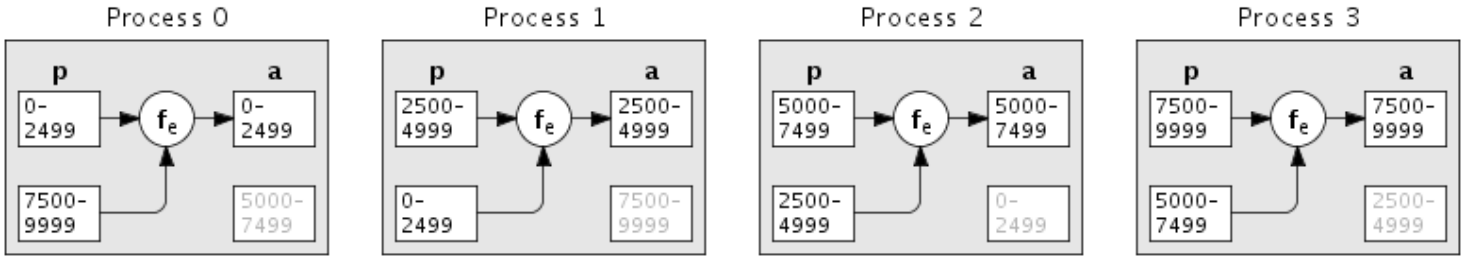
Swap the auxiliary buffers.
Compute the electrostatic forces between all particles in this process's slice and all particles in the just-received slice of positions,
acculumating the forces into the acceleration slice;
and, at the same time,
send the slice of positions in 1^{st} auxiliary buffer to the previous process and receive a slice of positions from the next process into 2^{nd} auxiliary buffer.

Overlapped Computation and Communication



Swap the auxiliary buffers.
Compute the electrostatic forces between all particles in this process's slice and all particles in the just-received slice of positions,
acculumating the forces into the acceleration slice;
and, at the same time,
send the slice of positions in 1^{st} auxiliary buffer to the previous process and receive a slice of positions from the next process into 2^{nd} auxiliary buffer.

Overlapped Computation and Communication



Swap the auxiliary buffers.
Compute the electrostatic forces between all particles in this process's slice and all particles in the just-received slice of positions,
acculumating the forces into the acceleration slice.

Non-Blocking Send-Receive

```
int toRank = (rank - 1 + size) % size;
int fromRank = (rank + 1) % size;
CommRequest req = new CommRequest();
world.sendReceive (toRank, outBuf, fromRank, inBuf, req);
// perform computation
req.waitForFinish();
```

AntiprotonClu3.java

code/AntiprotonClu3.java

AntiprotonClu3.java

code/AntiprotonClu3.java

Before looking at the running-time measurements for AntiprotonClu3, derive a model to predict the running time.

What will we need?

- a calculation time model (**same as** AntiprotonClu)
- a communication time model (**same as** AntiprotonClu2)

Calculation Time Model

Antiproton motion algorithm is $O(sn^2)$,
where n is the number of particles
and s is total number of steps ($s = snaps \cdot steps$).

$$T^{\bar{p}3}_{\mathbf{calc}}(s, n, 1) = asn^2$$

$$T^{\bar{p}3}_{\mathbf{calc}}(s, n, K) = asn^2 \frac{1}{K}$$

$$T^{\bar{p}3}_{\mathbf{calc}}(s, n, 1) = 2.04 \times 10^{-8} sn^2$$

$$T^{\bar{p}3}_{\mathbf{calc}}(s, n, K) = 2.04 \times 10^{-8} sn^2 \frac{1}{K}$$

Communication Time Model

- How many send-receives are performed? $s(K-1)$
- How many bits in each send-receives message? $128n\frac{1}{K}$

$$T^{\bar{p}3}_{\text{comm}}(s,n,K) = s(K-1)T_{\text{send-receive}}(128n\tfrac{1}{K},K)$$

$$T^{\bar{p}3}_{\text{comm}}(s,n,K) = s(K-1)(L+\tfrac{1}{B}128n\tfrac{1}{K})$$

$$T^{\bar{p}3}_{\text{comm}}(s,n,K) = s(K-1)(2.08\times10^{-4}+1.37\times10^{-7}n\tfrac{1}{K})$$

Running Time Model

$$T^{\bar{p}3}(s, n, K) = T^{\bar{p}3}_{\mathbf{calc}}(s, n, K) + T^{\bar{p}3}_{\mathbf{comm}}(s, n, K)$$

Running Time Model

$$T^{\bar{p}3}(s, n, K) = \max(T^{\bar{p}3}_{\mathbf{calc}}(s, n, K) , T^{\bar{p}3}_{\mathbf{comm}}(s, n, K))$$

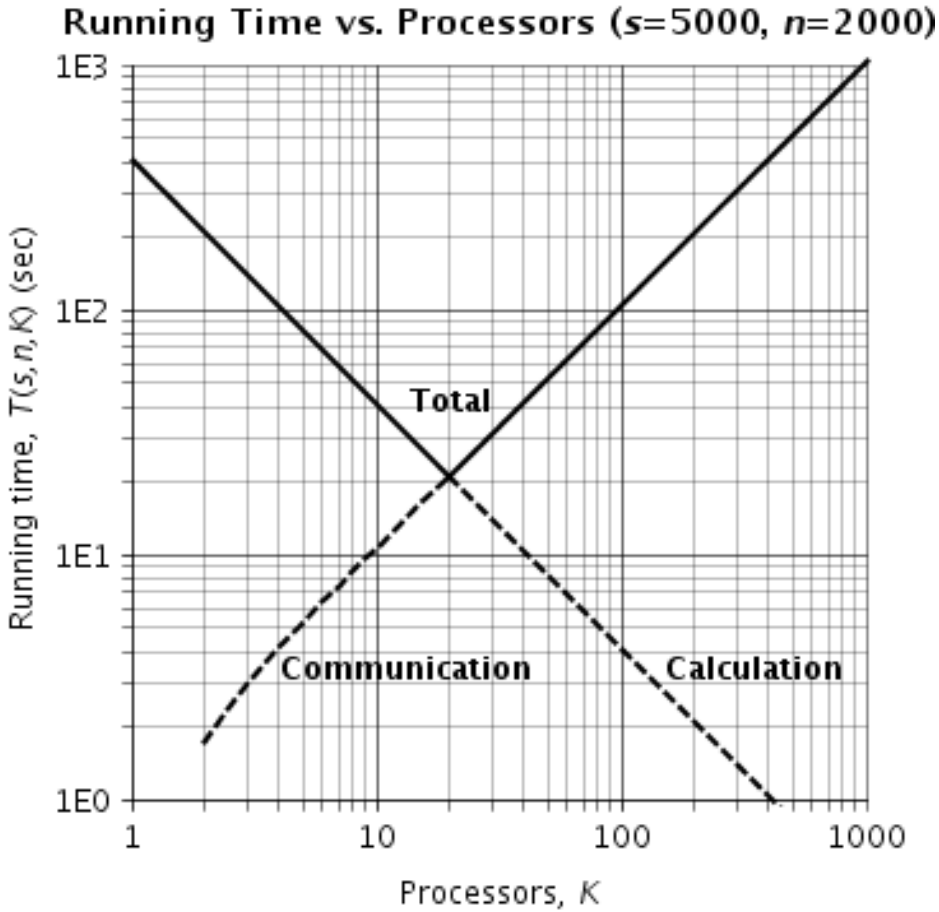
Running Time Model

$$T^{\bar{p}3}(s,n,K) = \max(\ T^{\bar{p}3}_{\text{calc}}(s,n,K)\ ,\ T^{\bar{p}3}_{\text{comm}}(s,n,K)\ )$$

$$T^{\bar{p}3}(s,n,K) = \max(\ asn^2\frac{1}{K}\ ,\ s(K-1)(L+\frac{1}{B}128n\frac{1}{K})\ )$$

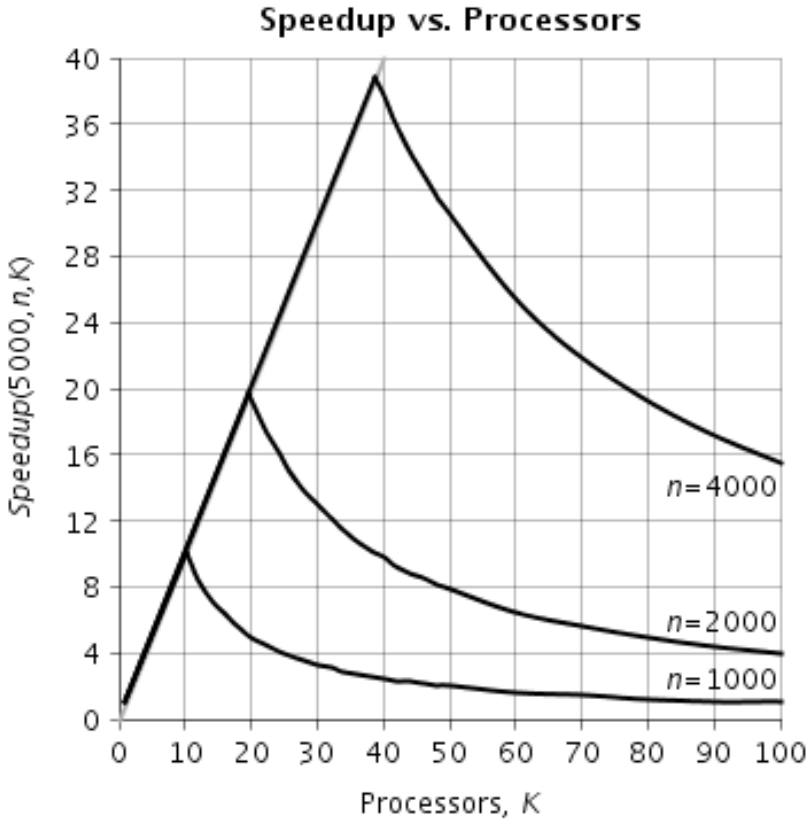
$$T^{\bar{p}3}(s,n,K) = \max(\ 2.04\times10^{-8}sn^2\frac{1}{K}\ ,\ s(K-1)(2.08\times10^{-4}+1.37\times10^{-7}n\frac{1}{K})\ )$$

Running Time Model



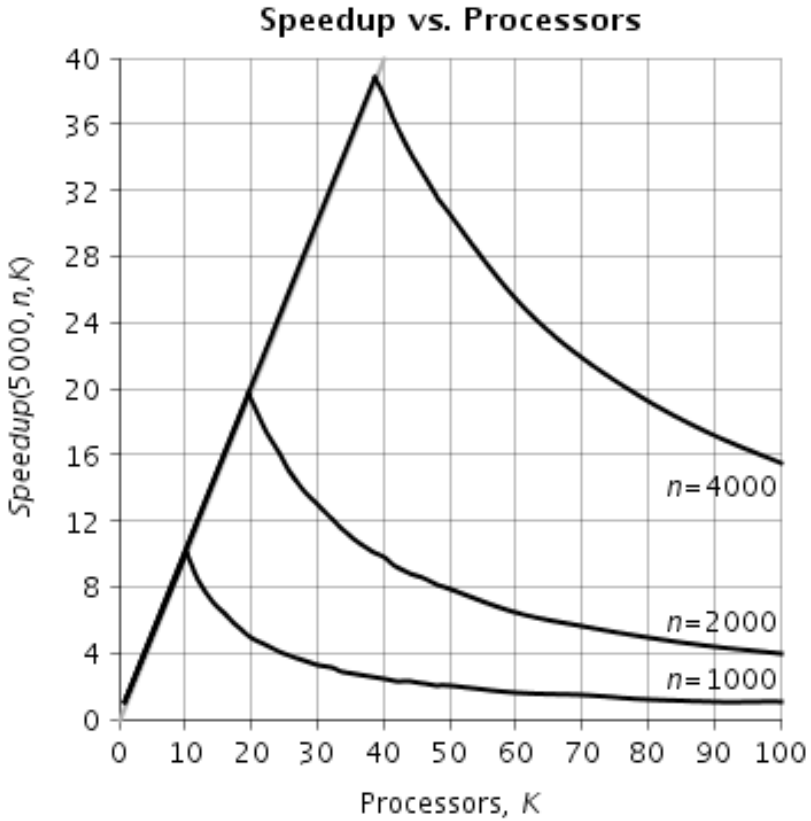
$$T^{\bar{p}3}(s,n,K) = \max(\ 2.04\times10^{-8} sn^2 \frac{1}{K}\ ,\ s(K-1)(2.08\times10^{-4} + 1.37\times10^{-7} n\frac{1}{K})\ )$$

Speedup Model



$$Speedup^{\bar{p}3}(s, n, K) = \frac{T^{\bar{p}3}(s, n, 1)}{T^{\bar{p}3}(s, n, K)}$$

Speedup Model



What is the maximum speedup that the program can acheive?

Speedup Model

What is the maximum speedup that the program can acheive?

Speedup Model

What is the maximum speedup that the program can acheive?

Find the value of K that results in the minimum running time.

Speedup Model

What is the maximum speedup that the program can acheive?

Find the value of K that results in the minimum running time.

Set $T^{\bar{p}3}_{\mathbf{calc}}(s, n, K)$ equal to $T^{\bar{p}3}_{\mathbf{comm}}(s, n, K)$ and solve for K.

Speedup Model

$$K_{\text{best}}^{\bar{p}3}(s, n) = \cdots$$

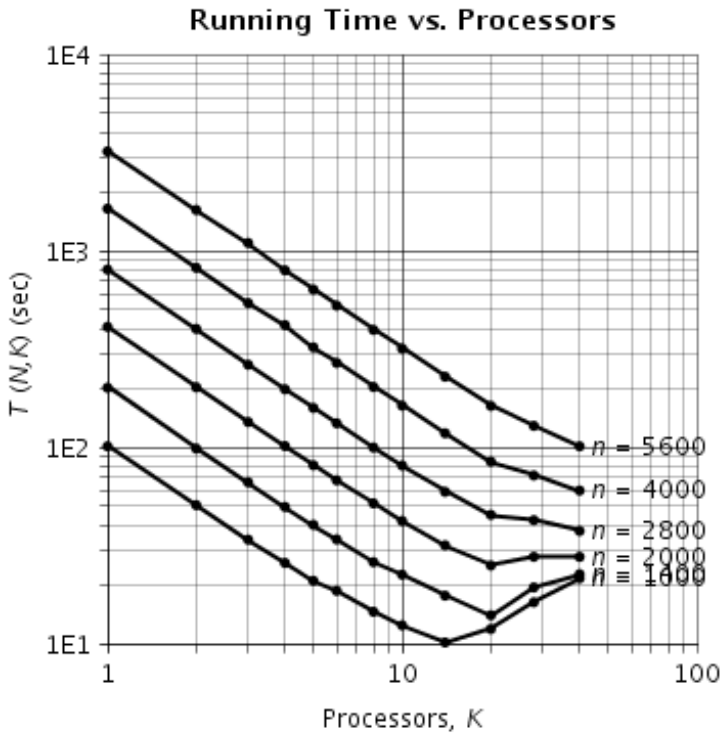
$$K_{\text{best}}^{\bar{p}3}(s, n) = \cdots$$

$$K_{\text{best}}^{\bar{p}3}(5000, 2000) \approx 20$$

$$Speedup^{\bar{p}3}(5000, 2000, 20) = 20$$

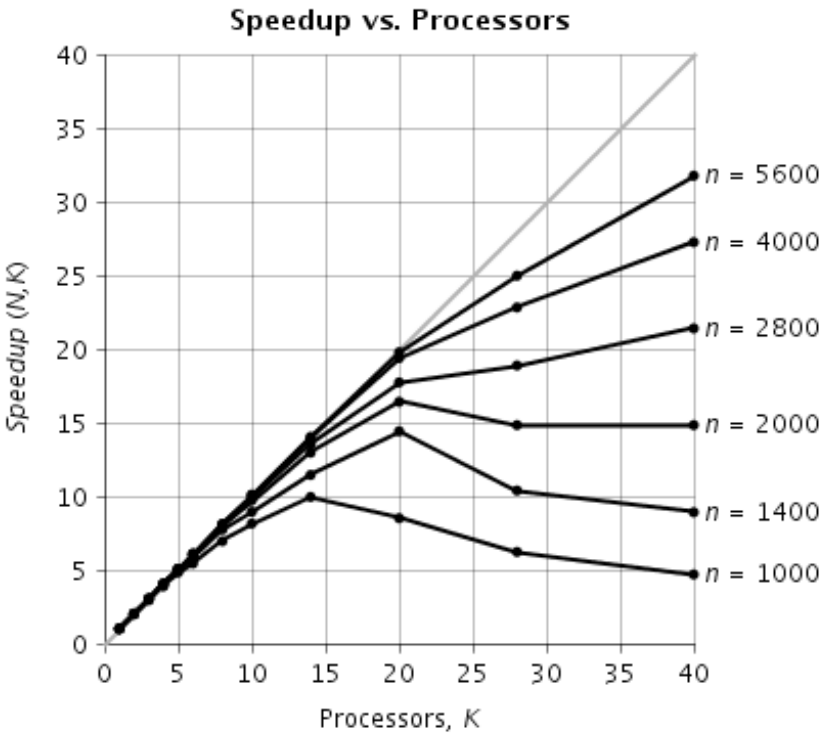
$$Eff^{\bar{p}3}(5000.2000, 20) = 1.000$$

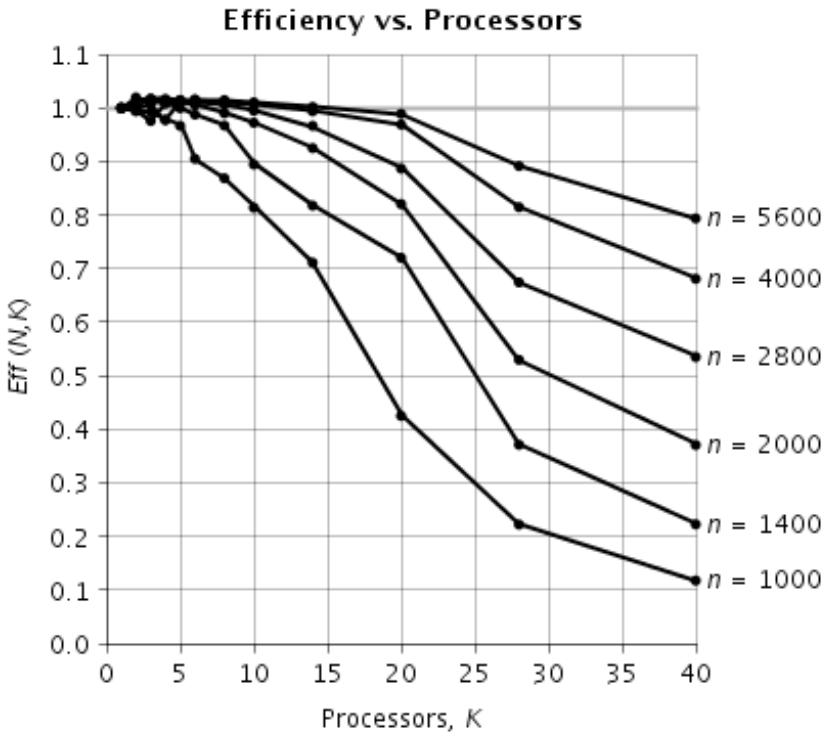
AntiprotonClu3 Running Time



$s = 5000$; n varying

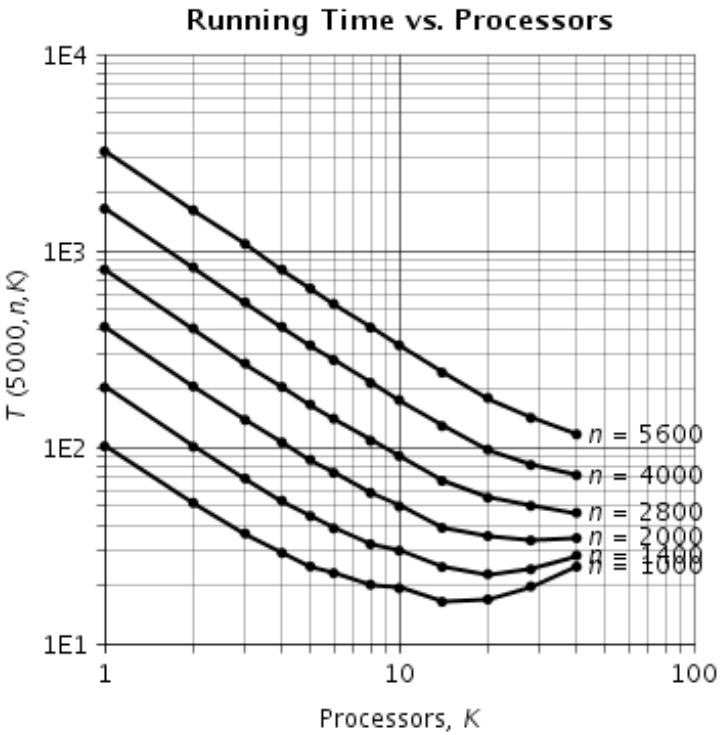
AntiprotonClu3 Speedup and Efficiency

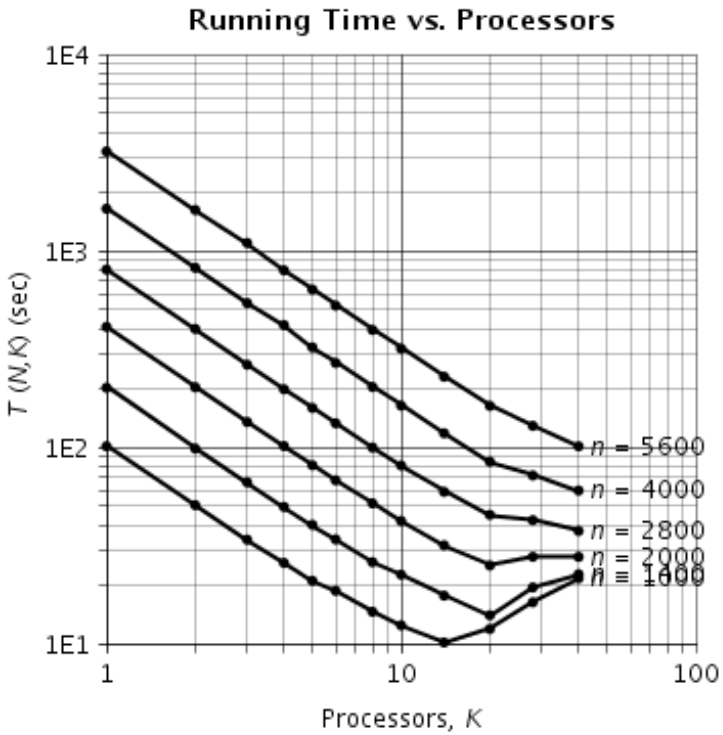




$s = 5000$; n varying

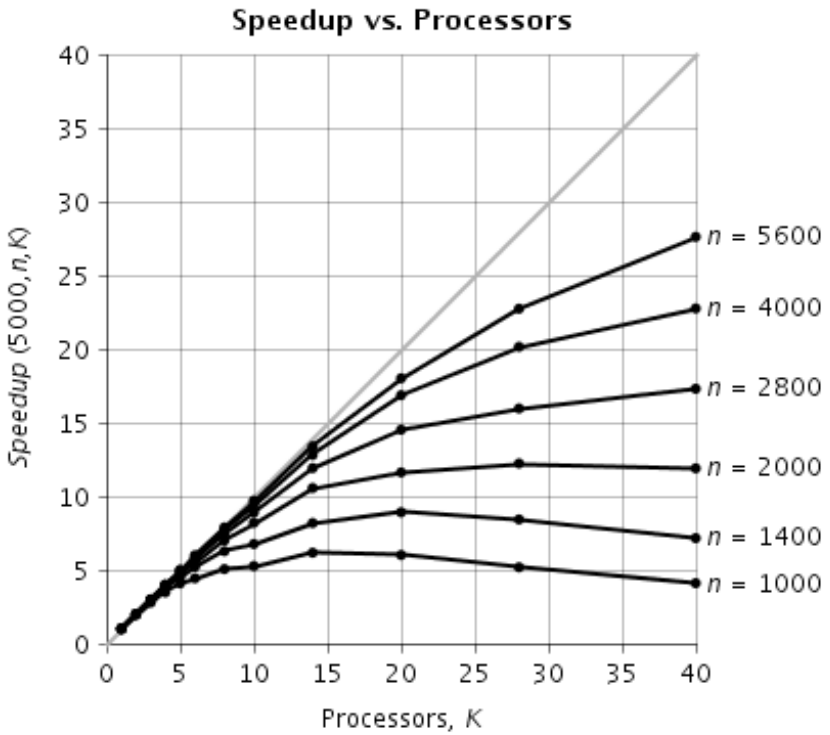
`AntiprotonClu` vs. `AntiprotonClu3` Running Time

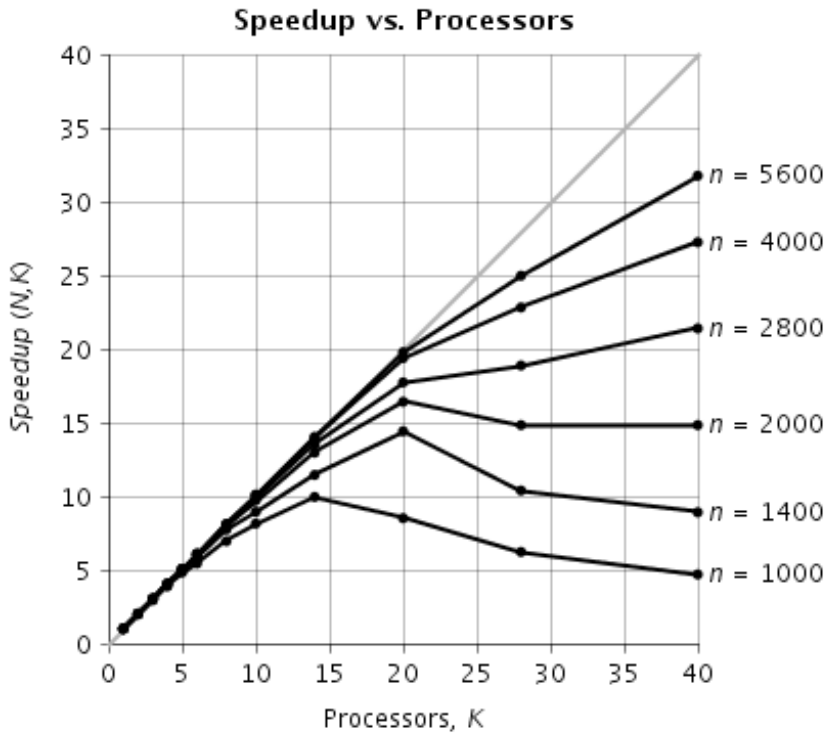




$s = 5000$; n varying

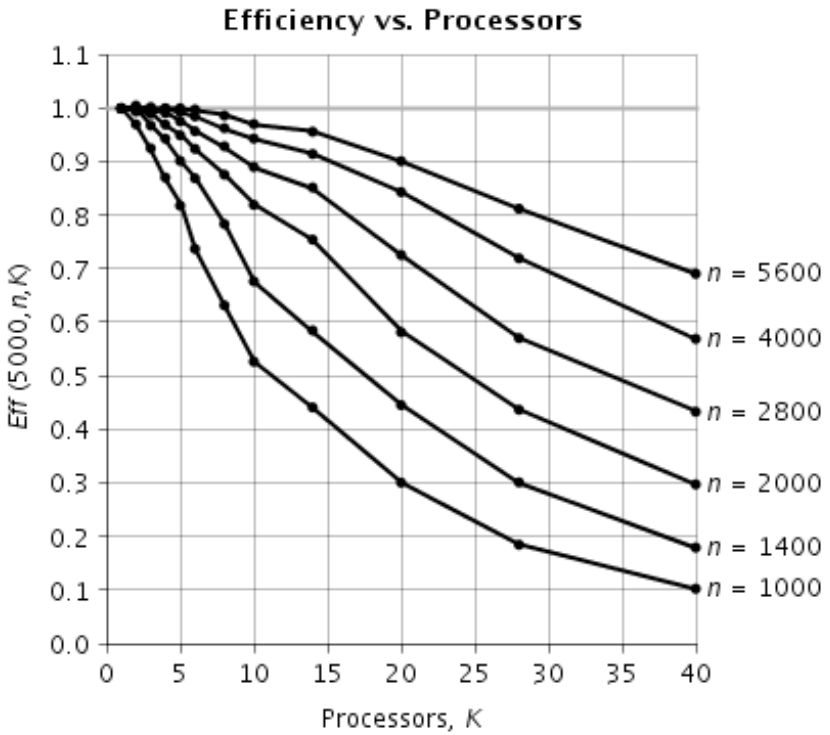
AntiprotonClu vs. AntiprotonClu3 Speedup

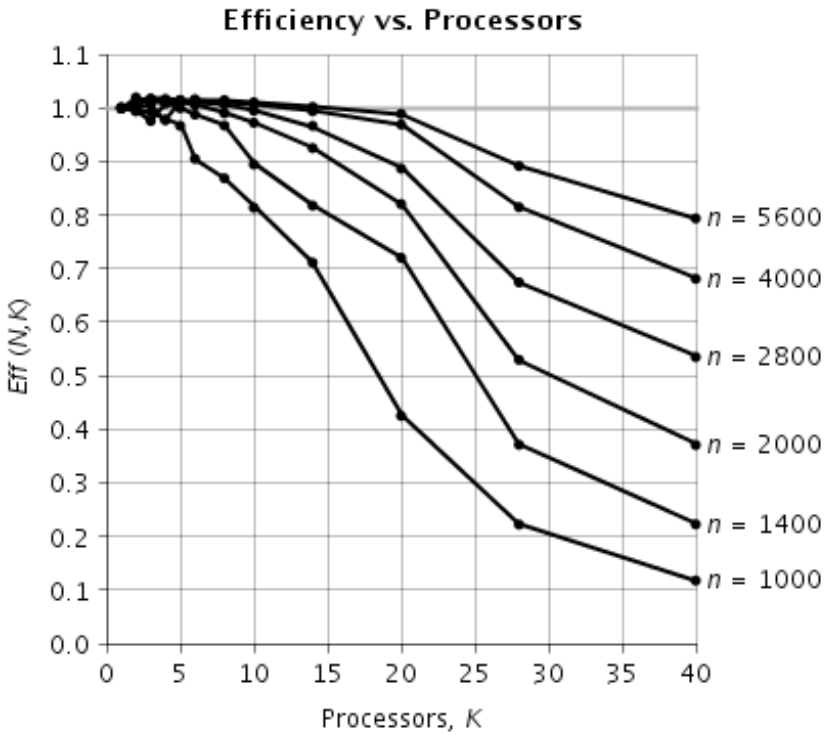




$s = 5000$; n varying

AntiprotonClu vs. AntiprotonClu3 Efficiency





$s = 5000$; n varying