Charon: linear algebra made easy

G. M. Crosswhite

Department of Physics
University of Washington

Thursday, April 16, 2009

My story

1. Quantum computing!
2. Can we build a reliable memory for quantum bits?
3. Numerical simulation of quantum systems
4. New algorithms for quantum simulations
5. Can they be made to scale?

My story

1. Quantum computing!
2. Can we build a reliable memory for quantum bits?
3. Numerical simulation of quantum systems
4. New algorithms for quantum simulations
5. Can they be made to scale?

My story

1. Quantum computing!
2. Can we build a reliable memory for quantum bits?
3. Numerical simulation of quantum systems
4. New algorithms for quantum simulations
5. Can they be made to scale?

My story

1. Quantum computing!
2. Can we build a reliable memory for quantum bits?
3. Numerical simulation of quantum systems
4. New algorithms for quantum simulations
5. Can they be made to scale?

My story

1. Quantum computing!
2. Can we build a reliable memory for quantum bits?
3. Numerical simulation of quantum systems
4. New algorithms for quantum simulations
5. Can they be made to scale?

My story

1. Quantum computing!
2. Can we build a reliable memory for quantum bits?
3. Numerical simulation of quantum systems
4. New algorithms for quantum simulations
5. Can they be made to scale?

Let's do it!

- Tools we need:
  - Tensor I/O
  - Tensor contractions
  - Singular Value Decompositions (SVDs)
  - Minimum eigenvalue solving
- Tools we have:
  - MPI Parallel I/O
  - Global Arrays
  - SciLAPACK
  - ARPACK
- Problems:
  - Incomplete!
  - Cumbersome to use!
    - Lots of boilerplate and plumbing code needed
    - Synchronous communication model

Let's do it!

- Tools we need:
  - Tensor I/O
  - Tensor contractions
  - Singular Value Decompositions (SVDs)
  - Minimum eigenvalue solving
- Tools we have:
  - MPI Parallel I/O
  - Global Arrays
  - SciLAPACK
  - ARPACK
- Problems:
  - Incomplete!
  - Cumbersome to use!
    - Lots of boilerplate and plumbing code needed
    - Synchronous communication model

Let's do it!

- Tools we need:
  - Tensor I/O
  - Tensor contractions
  - Singular Value Decompositions (SVDs)
  - Minimum eigenvalue solving
- Tools we have:
  - MPI Parallel I/O
  - Global Arrays
  - SciLAPACK
  - ARPACK
- Problems:
  - Incomplete!
  - Cumbersome to use!
    - Lots of boilerplate and plumbing code needed
    - Synchronous communication model

Let's do it!

- Tools we need:
  - Tensor I/O
  - Tensor contractions
  - Singular Value Decompositions (SVDs)
  - Minimum eigenvalue solving
- Tools we have:
  - MPI Parallel I/O
  - Global Arrays
  - SciLAPACK
  - ARPACK
- Problems:
  - Incomplete!
  - Cumbersome to use!
    - Lots of boilerplate and plumbing code needed
    - Synchronous communication model

The Vision

A parallel linear-algebra intensive code should not be more complicated than the algorithm being implemented.

A Simple Example

1. Read in an array
2. Increment all entries in the array by 1
3. Sum all of the entries in the array
4. Divide all entries in the array by the sum
5. Write out the array

A Simple Example

```
DistributedArray<float,1> A(16,1<<20);
A.loadFrom("infile");
A += 1;
float s = A.sum();
A /= s;
A.writeTo("outfile");
```

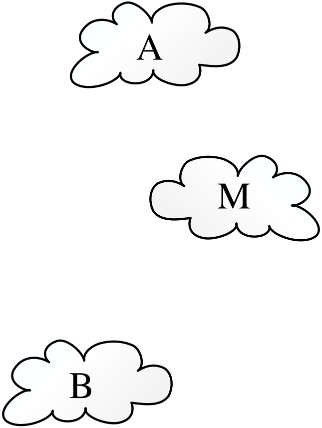
A More Complicated Example

1. Read in matrices A and B
2. Invert A and B
3. Multiply them together to form M
4. Break M apart back into A and B using a SVD
5. Invert A and B again
6. Save A and B

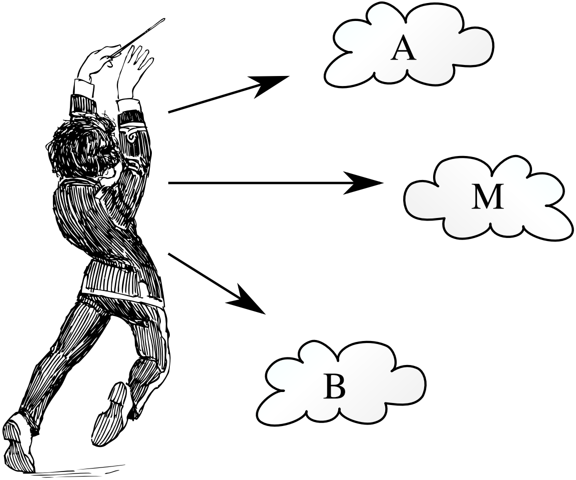
A More Complicated Example

```
DistributedArray<float,2> A(8,1024,1024),
    B(8,1024,1024), M(16,1024,1024);
DistributedArray<float,1> Sigma(16,1024);
A.loadFrom("A.in"); B.loadFrom("B.in");
inv(A); inv(B);
matmul(A,B,M);
svd(S,A,Sigma,B);
inv(A); inv(B);
A.writeTo("A.out"); B.writeTo("B.out");
```

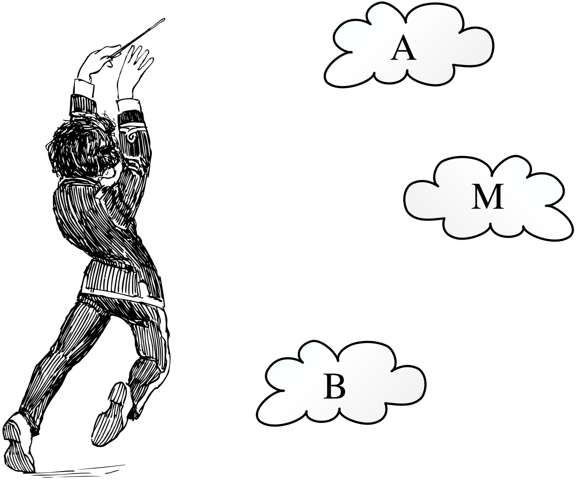
Maestro, please!



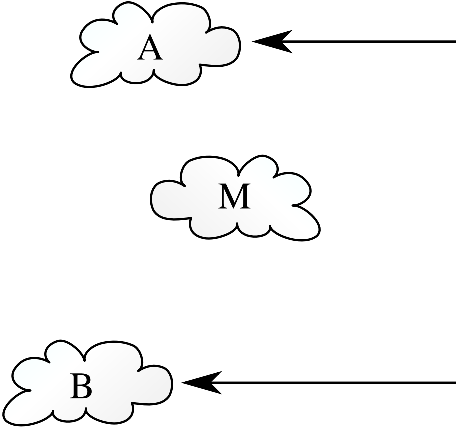
Maestro, please!



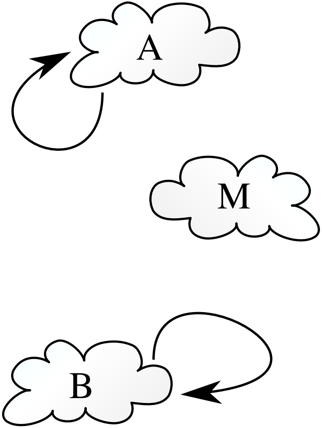
Maestro, please!



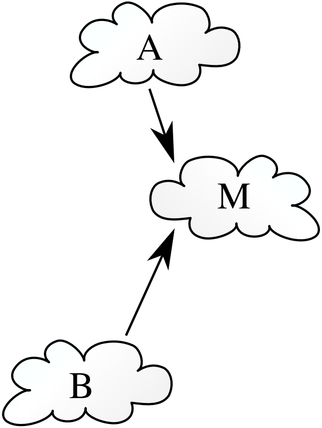
Maestro, please!



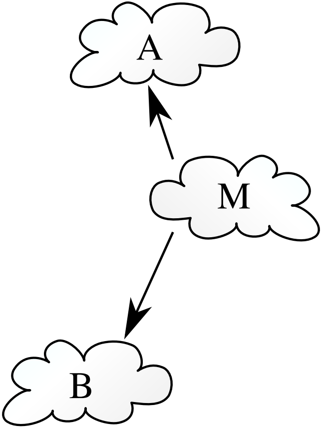
Maestro, please!



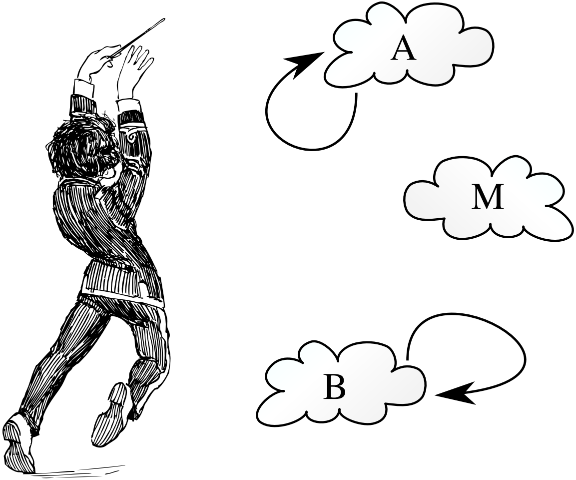
Maestro, please!



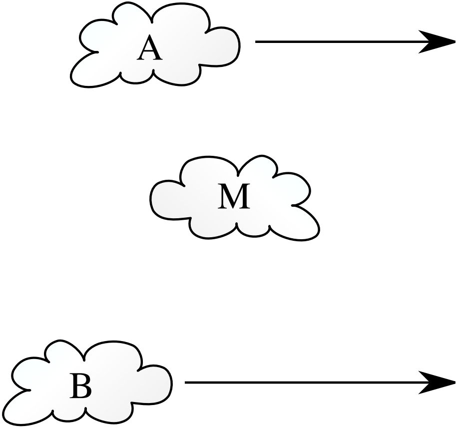
Maestro, please!



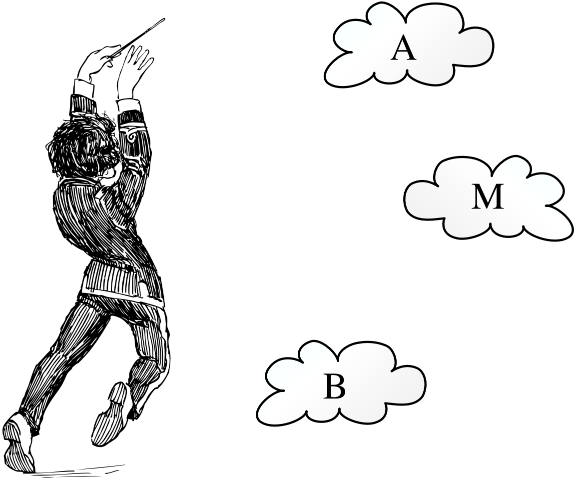
Maestro, please!



Maestro, please!



Maestro, please!



The Recipe

Ingrediants:

- Asynchronous communication
- Master/Slave architecture
- Explicit ordering and data dependencies

The result:

- Local coordination of task scheduling
    - (Caveat: Central decisions)
- Automatic parallelization of parallel tasks

(Effectively building and walking a DAG.)

The Recipe

Ingrediants:

- Asynchronous communication
- Master/Slave architecture
- Explicit ordering and data dependencies

The result:

- Local coordination of task scheduling
  - (Caveat: Central decisions)
- Automatic parallelization of parallel tasks

(Effectively building and walking a DAG.)

The Recipe

Ingrediants:

- Asynchronous communication
- Master/Slave architecture
- Explicit ordering and data dependencies

The result:

- Local coordination of task scheduling
  - (Caveat: Central decisions)
- Automatic parallelization of parallel tasks

(Effectively building and walking a DAG.)

The Recipe

Ingrediants:

- Asynchronous communication
- Master/Slave architecture
- Explicit ordering and data dependencies

The result:

- Local coordination of task scheduling
  - (Caveat: Central decisions)
- Automatic parallelization of parallel tasks

(Effectively building and walking a DAG.)

Why Charm++?

- Asynchronous communication model
- AMPI provides “virtualization” MPI libraries
- Emphasis on higher-level programming

Why Charm++?

- Asynchronous communication model
- AMPI provides “virtualization” MPI libraries
- Emphasis on higher-level programming

Why Charm++?

- Asynchronous communication model
- AMPI provides “virtualization” MPI libraries
- Emphasis on higher-level programming

Why Charm++?

- Asynchronous communication model
- AMPI provides “virtualization” MPI libraries
- Emphasis on higher-level programming

The Components

- Array Master/Slave Controller
  - Distributed Array
  - Block-cyclic Array
- Operations
- AMPI Master/Slave Controller
  - BLACS Grid Master/Slave (interface to BLACS)
  - Block IO Master/Slave (interface to ROMIO)
- Matrix multiplier

The Components

- Array Master/Slave Controller
  - Distributed Array
  - Block-cyclic Array
- Operations
- AMPI Master/Slave Controller
  - BLACS Grid Master/Slave (interface to BLACS)
  - Block IO Master/Slave (interface to ROMIO)
- Matrix multiplier

The Components

- Array Master/Slave Controller
  - Distributed Array
  - Block-cyclic Array
- Operations
- AMPI Master/Slave Controller
  - BLACS Grid Master/Slave (interface to BLACS)
  - Block IO Master/Slave (interface to ROMIO)
- Matrix multiplier

The Components

- Array Master/Slave Controller
  - Distributed Array
  - Block-cyclic Array
- Operations
- AMPI Master/Slave Controller
  - BLACS Grid Master/Slave (interface to BLACS)
  - Block IO Master/Slave (interface to ROMIO)
- Matrix multiplier

The Components

- Array Master/Slave Controller
  - Distributed Array
  - Block-cyclic Array
- Operations
- AMPI Master/Slave Controller
  - BLACS Grid Master/Slave (interface to BLACS)
  - Block IO Master/Slave (interface to ROMIO)
- Matrix multiplier

The Components

- Array Master/Slave Controller $\longrightarrow$ ordered
  - Distributed Array
  - Block-cyclic Array
- Operations
- AMPI Master/Slave Controller $\longrightarrow$ coordinated
  - BLACS Grid Master/Slave (interface to BLACS)
  - Block IO Master/Slave (interface to ROMIO)
- Matrix multiplier

Array Master/Slave Architecture

Question: How do we enforce ordering of operations?

Answer: Operation counters + slave priority queues

Question: How do we implement the counter?

- Global counter for all slaves
  - Commands sent to single slaves waste bandwidth!
- Separate counter for each slave
  - Huge table needed to track counters!
  - Global operations can no longer be broadcasted!

Array Master/Slave Architecture

Question: How do we enforce ordering of operations?
Answer: Operation counters + slave priority queues

Question: How do we implement the counter?

- Global counter for all slaves
  - Commands sent to single slaves waste bandwidth!
- Separate counter for each slave
  - Huge table needed to track counters!
  - Global operations can no longer be broadcasted!

Array Master/Slave Architecture

Question: How do we enforce ordering of operations?
Answer: Operation counters + slave priority queues

Question: How do we implement the counter?

- Global counter for all slaves
    - Commands sent to single slaves waste bandwidth!
- Separate counter for each slave
    - Huge table needed to track counters!
    - Global operations can no longer be broadcasted!

Array Master/Slave Architecture

Question: How do we enforce ordering of operations?
Answer: Operation counters + slave priority queues

Question: How do we implement the counter?

- Global counter for all slaves
    - Commands sent to single slaves waste bandwidth!
- Separate counter for each slave
    - Huge table needed to track counters!
    - Global operations can no longer be broadcasted!

Array Master/Slave Architecture

Question: How do we enforce ordering of operations?
Answer: Operation counters + slave priority queues

Question: How do we implement the counter?

- Global counter for all slaves
  - Commands sent to single slaves waste bandwidth!
- Separate counter for each slave
  - Huge table needed to track counters!
  - Global operations can no longer be broadcasted!

Array Master/Slave Architecture

Question: How do we enforce ordering of operations?
Answer: Operation counters + slave priority queues

Question: How do we implement the counter?

- Global counter for all slaves
  - Commands sent to single slaves waste bandwidth!
- Separate counter for each slave
  - Huge table needed to track counters!
  - Global operations can no longer be broadcasted!

Array Master/Slave Architecture

Question: How do we enforce ordering of operations?
Answer: Operation counters + slave priority queues

Question: How do we implement the counter?

- Global counter for all slaves
  - Commands sent to single slaves waste bandwidth!
- Separate counter for each slave
  - Huge table needed to track counters!
  - Global operations can no longer be broadcasted!

Array Master/Slave Architecture

Question: How do we enforce ordering of operations?
Answer: Operation counters + slave priority queues

Question: How do we implement the counter?

- Global counter for all slaves
  - Commands sent to single slaves waste bandwidth!
- Separate counter for each slave
  - Huge table needed to track counters!
  - Global operations can no longer be broadcasted!

Array Master/Slave Architecture

My solution: Global counter with “stealing”

- Every operation increments the global counter
- Single slaves may “steal” a value of the counter
- Broadcasts to all slaves contain a list of who stole the counter since the last broadcast. This way every slave knows which values they should skip and which they need to wait for.

Example:

1. Broadcast 1 $\rightarrow$ 1, []
2. Pointcast 2 to A
3. Pointcast 3 to B
4. Broadcast 1 $\rightarrow$ 4, [A,B]

Array Master/Slave Architecture

My solution: Global counter with “stealing”

- Every operation increments the global counter
- Single slaves may “steal” a value of the counter
- Broadcasts to all slaves contain a list of who stole the counter since the last broadcast. This way every slave knows which values they should skip and which they need to wait for.

Example:

1. Broadcast 1 $\rightarrow$ 1, []
2. Pointcast 2 to A
3. Pointcast 3 to B
4. Broadcast 1 $\rightarrow$ 4, [A,B]

Array Master/Slave Architecture

My solution: Global counter with “stealing”

- Every operation increments the global counter
- Single slaves may “steal” a value of the counter
- Broadcasts to all slaves contain a list of who stole the counter since the last broadcast. This way every slave knows which values they should skip and which they need to wait for.

Example:

1. Broadcast 1 $\rightarrow$ 1, []
2. Pointcast 2 to A
3. Pointcast 3 to B
4. Broadcast 1 $\rightarrow$ 4, [A,B]

Array Master/Slave Architecture

My solution: Global counter with “stealing”

- Every operation increments the global counter
- Single slaves may “steal” a value of the counter
- Broadcasts to all slaves contain a list of who stole the counter since the last broadcast. This way every slave knows which values they should skip and which they need to wait for.

Example:

1. Broadcast 1 $\rightarrow$ 1, []
2. Pointcast 2 to A
3. Pointcast 3 to B
4. Broadcast 1 $\rightarrow$ 4, [A,B]

Array Master/Slave Architecture

My solution: Global counter with "stealing"

- Every operation increments the global counter
- Single slaves may "steal" a value of the counter
- Broadcasts to all slaves contain a list of who stole the counter since the last broadcast. This way every slave knows which values they should skip and which they need to wait for.

Example:

1. Broadcast 1 $\rightarrow$ 1, []
2. Pointcast 2 to A
3. Pointcast 3 to B
4. Broadcast 1 $\rightarrow$ 4, [A,B]

DistributedArray

- Templated on *type* and *dimension*
- Master:
  - Operation ordering
  - Addressing – map from coordinates to slave number
- Slaves: (1D array)
  - Operation execution
  - Local data stored using Blitz++, a high level array class templated on *type* and *dimension*

DistributedArray

- Templated on *type* and *dimension*
- Master:
  - Operation ordering
  - Addressing – map from coordinates to slave number
- Slaves: (1D array)
  - Operation execution
  - Local data stored using Blitz++, a high level array class templated on *type* and *dimension*

DistributedArray

- Templated on *type* and *dimension*
- Master:
  - Operation ordering
  - Addressing – map from coordinates to slave number
- Slaves: (1D array)
  - Operation execution
  - Local data stored using Blitz++, a high level array class templated on *type* and *dimension*

DistributedArray

- Templated on *type* and *dimension*
- Master:
  - Operation ordering
  - Addressing – map from coordinates to slave number
- Slaves: (1D array)
  - Operation execution
  - Local data stored using Blitz++, a high level array class templated on *type* and *dimension*

Operations

- Whole-array transformations
  - add/subtract/divide/multiply by a constant
  - sine, cosine, absolute value
  - randomization
  - etc.
- Single-element transformations
- Array reductions (sum, product, etc.)

Stupidly easy to implement more:

- Add to one of my switch statements
- Subclass `Operation`

Templates are your friend!

- Don't need to case over the type of the array.
- Not limited to numeric types / operations!

Operations

- Whole-array transformations
  - add/subtract/divide/multiply by a constant
  - sine, cosine, absolute value
  - randomization
  - etc.
- Single-element transformations
- Array reductions (sum, product, etc.)

Stupidly easy to implement more:

- Add to one of my switch statements
- Subclass `Operation`

Templates are your friend!

- Don't need to case over the type of the array.
- Not limited to numeric types / operations!

Operations

- Whole-array transformations
  - add/subtract/divide/multiply by a constant
  - sine, cosine, absolute value
  - randomization
  - etc.
- Single-element transformations
- Array reductions (sum, product, etc.)

Stupidly easy to implement more:

- Add to one of my switch statements
- Subclass `Operation`

Templates are your friend!

- Don't need to case over the type of the array.
- Not limited to numeric types / operations!

Example

```
DistributedArray<int,3> vec(6,2,3,4);
Array<int,3> x(2,3,4);

x = 1, 2, 3, 4,
    5, 6, 7, 8,
    9,10,11,12,

    -1,-2,-3,-5,
    -7,-9,-11,-13,
    -17,-19,-23,-29,
    -31,-37,-41,-43;

vec = x;
```

Example

```
vec++;
vec *= -1;

cout << "0,0,0 = " << (int)vec(0,0,0) << endl;

for(int i = 0; i < 2; i++)
    for(int j = 0; j < 3; j++)
        for(int k = 0; k < 4; k += 2)
            vec(i,j,k) *= -1;

vec.abs();

vec.gatherInto(x);
```

AMPI Controller

- Purpose: To allow access to libraries written in MPI
  - BLACS – opens the door to parallel linear algebra libraries such as ScaLAPACK
  - ROMIO – parallel I/O
- Contains no data itself
- Requires coordination but not ordering
- Uses TCharm to provide the virtual MPI layer
  - Slaves inherit from TCharm
  - Constructor launches the TCharm thread
  - TCharm thread pulls operations from a “ready” queue until none are left, then it goes to sleep
  - Slave chare wakes up the thread when a new operation is ready to be run

AMPI Controller

- Purpose: To allow access to libraries written in MPI
  - BLACS – opens the door to parallel linear algebra libraries such as ScaLAPACK
  - ROMIO – parallel I/O
- Contains no data itself
- Requires coordination but not ordering
- Uses TCharm to provide the virtual MPI layer
  - Slaves inherit from TCharm
  - Constructor launches the TCharm thread
  - TCharm thread pulls operations from a “ready” queue until none are left, then it goes to sleep
  - Slave chare wakes up the thread when a new operation is ready to be run

AMPI Controller

- Purpose: To allow access to libraries written in MPI
  - BLACS – opens the door to parallel linear algebra libraries such as ScaLAPACK
  - ROMIO – parallel I/O
- Contains no data itself
- Requires coordination but not ordering
- Uses TCharm to provide the virtual MPI layer
  - Slaves inherit from TCharm
  - Constructor launches the TCharm thread
  - TCharm thread pulls operations from a “ready” queue until none are left, then it goes to sleep
  - Slave chare wakes up the thread when a new operation is ready to be run

AMPI Controller

- Purpose: To allow access to libraries written in MPI
  - BLACS – opens the door to parallel linear algebra libraries such as ScaLAPACK
  - ROMIO – parallel I/O
- Contains no data itself
- Requires coordination but not ordering
- Uses TCharm to provide the virtual MPI layer
  - Slaves inherit from TCharm
  - Constructor launches the TCharm thread
  - TCharm thread pulls operations from a “ready” queue until none are left, then it goes to sleep
  - Slave chare wakes up the thread when a new operation is ready to be run

AMPI Controller

- Purpose: To allow access to libraries written in MPI
  - BLACS – opens the door to parallel linear algebra libraries such as ScaLAPACK
  - ROMIO – parallel I/O
- Contains no data itself
- Requires coordination but not ordering
- Uses TCharm to provide the virtual MPI layer
  - Slaves inherit from TCharm
  - Constructor launches the TCharm thread
  - TCharm thread pulls operations from a “ready” queue until none are left, then it goes to sleep
  - Slave chare wakes up the thread when a new operation is ready to be run

Privateer

Problem: Most C libraries are not designed to be multi-threaded! They assume that they have exclusive access to their global/static variables.

Solution: Replace all references to global/static variables with pointers into a thread-local structure.

Privateer accomplishes this, and is designed to work on arbitrary C code; it uses a Converse thread-private variable to store a pointer to the global variable table for the thread.

`http://launchpad.net/privateer`

Privateer

Problem: Most C libraries are not designed to be multi-threaded! They assume that they have exclusive access to their global/static variables.

Solution: Replace all references to global/static variables with pointers into a thread-local structure.

Privateer accomplishes this, and is designed to work on arbitrary C code; it uses a Converse thread-private variable to store a pointer to the global variable table for the thread.

`http://launchpad.net/privateer`

Privateer

Problem: Most C libraries are not designed to be multi-threaded! They assume that they have exclusive access to their global/static variables.

Solution: Replace all references to global/static variables with pointers into a thread-local structure.

Privateer accomplishes this, and is designed to work on arbitrary C code; it uses a Converse thread-private variable to store a pointer to the global variable table for the thread.

`http://launchpad.net/privateer`

Matrix Multiplication

$$\sum_j \mathbf{A}_{ij} \cdot \mathbf{B}_{jk} = \mathbf{C}_{ik}$$

Chare (i, j, k) computes $\mathbf{A}_{ij} \cdot \mathbf{B}_{jk}$, and then contributes to a sum reduction on the section $(i, :, k)$.

Chunks sent in an `ArrayMessage` to minimize copying.

Each chare *only* needs to know

- Reduction section
- Result callback

Matrix Multiplication

$$\sum_j \mathbf{A}_{ij} \cdot \mathbf{B}_{jk} = \mathbf{C}_{ik}$$

Chare (i, j, k) computes $\mathbf{A}_{ij} \cdot \mathbf{B}_{jk}$, and then contributes to a sum reduction on the section $(i, :, k)$.

Chunks sent in an `ArrayMessage` to minimize copying.

Each chare *only* needs to know

- Reduction section
- Result callback

Matrix Multiplication

$$\sum_j \mathbf{A}_{ij} \cdot \mathbf{B}_{jk} = \mathbf{C}_{ik}$$

Chare (i, j, k) computes $\mathbf{A}_{ij} \cdot \mathbf{B}_{jk}$, and then contributes to a sum reduction on the section $(i, :, k)$.

Chunks sent in an `ArrayMessage` to minimize copying.

Each chare *only* needs to know

- Reduction section
- Result callback

Matrix Multiplication

$$\sum_j \mathbf{A}_{ij} \cdot \mathbf{B}_{jk} = \mathbf{C}_{ik}$$

Chare (i, j, k) computes $\mathbf{A}_{ij} \cdot \mathbf{B}_{jk}$, and then contributes to a sum reduction on the section $(i, :, k)$.

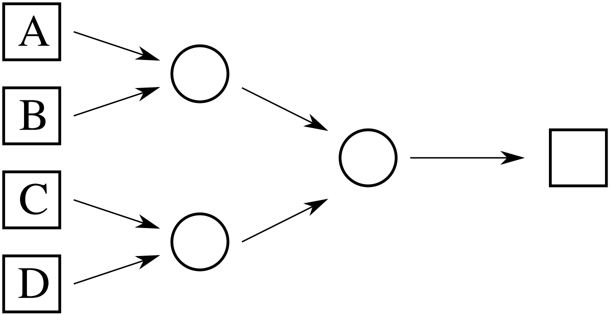
Chunks sent in an `ArrayMessage` to minimize copying.

Each chare *only* needs to know

- Reduction section
- Result callback

Matrix Multiplication

$$\mathbf{A} \cdot \mathbf{B} \cdot \mathbf{C} \cdot \mathbf{D} = (\mathbf{A} \cdot \mathbf{B}) \cdot (\mathbf{C} \cdot \mathbf{D})$$



Closing

Status:

- Basic infrastructure largely complete; lots of simple operations could easily be implemented
- Still wrestling with getting libraries to work under AMPI
  - (Global variable privatization problem has been solved by `Privateer`.)
- Need to profile matrix multiplication algorithm and extend to implement tensor contractions

Repositories Online:

- `http://launchpad.net/privateer`
- `http://launchpad.net/charon`

Please let me know what you think and/or want!

Closing

Status:

- Basic infrastructure largely complete; lots of simple operations could easily be implemented
- Still wrestling with getting libraries to work under AMPI
  - (Global variable privatization problem has been solved by `Privateer`.)
- Need to profile matrix multiplication algorithm and extend to implement tensor contractions

Repositories Online:

- `http://launchpad.net/privateer`
- `http://launchpad.net/charon`

Please let me know what you think and/or want!

Closing

Status:

- Basic infrastructure largely complete; lots of simple operations could easily be implemented
- Still wrestling with getting libraries to work under AMPI
  - (Global variable privatization problem has been solved by `Privateer`.)
- Need to profile matrix multiplication algorithm and extend to implement tensor contractions

Repositories Online:

- `http://launchpad.net/privateer`
- `http://launchpad.net/charon`

Please let me know what you think and/or want!

Closing

Status:

- Basic infrastructure largely complete; lots of simple operations could easily be implemented
- Still wrestling with getting libraries to work under AMPI
  - (Global variable privatization problem has been solved by `Privateer`.)
- Need to profile matrix multiplication algorithm and extend to implement tensor contractions

Repositories Online:

- `http://launchpad.net/privateer`
- `http://launchpad.net/charon`

Please let me know what you think and/or want!

Closing

Status:

- Basic infrastructure largely complete; lots of simple operations could easily be implemented
- Still wrestling with getting libraries to work under AMPI
  - (Global variable privatization problem has been solved by `Privateer`.)
- Need to profile matrix multiplication algorithm and extend to implement tensor contractions

Repositories Online:

- `http://launchpad.net/privateer`
- `http://launchpad.net/charon`

Please let me know what you think and/or want!

Closing

Status:

- Basic infrastructure largely complete; lots of simple operations could easily be implemented
- Still wrestling with getting libraries to work under AMPI
  - (Global variable privatization problem has been solved by `Privateer`.)
- Need to profile matrix multiplication algorithm and extend to implement tensor contractions

Repositories Online:

- `http://launchpad.net/privateer`
- `http://launchpad.net/charon`

Please let me know what you think and/or want!