A Grand Unified Theory of MxN Parallel Data Redistribution
(a.k.a. throw another speed-o on the fire… 😊)

June 14, 2001

James “Jeeembo” Kohl
Oak Ridge National Laboratory
(with ideas and work stolen from Alan, David, Gary, Kate, Scott & Others!)

Research supported by the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing Research, U.S. Department of Energy, under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.
MxN Internal Structure

MxN Data Exchange and Synchronization Interface

Parallel Data Mapping and Communication

Data Decomposition Specification

Data Distribution System

Local Data
CUMULVS (ORNL):

- MxN Exchange and Sync Interface (persistent channel)
- Parallel Data Map & Comm
- Data Decomp Specification

(PVM, MPI)

“Eyes” Component
(Out-of-Band Viewer Connection, Wholesale Use of Existing Tools)
MxN Demo – SC00

CUMULVS (ORNL):

- MxN Exchange and Sync Interface (persistent channel)
- Parallel Data Map & Comm
- Data Decomp Specification

(PVM, MPI)

LOCAL DATA

“Eyes” Component
(Still Out-of-Band Viewer Connection… SPMD Model)

Data Holder Prototype Component

Migration Towards True MxN Component Organization

ORNL/Kohl 2001
# MxN Technology Sources

<table>
<thead>
<tr>
<th>Source</th>
<th>MxN Exchange and Sync Interface (persistent channel)</th>
<th>Parallel Data Map &amp; Comm</th>
<th>Data Decomp Specification</th>
<th>Global Arrays (PNNL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUMULVS (ORNL)</td>
<td></td>
<td></td>
<td></td>
<td>LOCAL DATA</td>
</tr>
<tr>
<td>PAWS (LANL)</td>
<td></td>
<td></td>
<td></td>
<td>MPI</td>
</tr>
<tr>
<td>PADRE (LANL)</td>
<td></td>
<td></td>
<td></td>
<td>KeLP (UCSD)</td>
</tr>
<tr>
<td>Meta-Chaos (U Maryland)</td>
<td></td>
<td></td>
<td></td>
<td>PETSc (ANL)</td>
</tr>
</tbody>
</table>

- Data Distribution System
- Other Data Dist Tools…

**LOCAL DATA**

**GLOBAL ARRAY**

- PVM
- MPI

**OTHER DATA**
MxN Technology Integration

- MxN Exchange and Sync Interface (persistent channel and point-to-point)
- Parallel Data Map & Comm
- Data Decomp Specification
- Data Distribution System

CCA MxN Specification

CCA MxN Component

CCA Data Object

New & Existing Tools, including:
- Global Arrays (PNNL),
- KeLP (UCSD), PETSc (ANL)…

CUMULVS, PAWS, Meta-Chaos…

plus PADRE…

LOCAL DATA
Integrated MxN Port Interface

class MxN : Port {
    virtual void registerData( Data A );
    virtual void unregisterData( Data A );
    // (Data Properties: synch)
    virtual int getData( Data A, Data B );
    virtual int getDataNonBlocking( A, B );
    virtual void waitData( A, B );
    virtual void releaseData( A, B );
    // Port Properties: freq, init_synch, run_synch…
}
MxN Port Properties

• Freq ~ Data Exchange Frequency
  – 0: one-shot, send/receive model (PAWS)
  – > 0: persistent conn, loose synch (CUMULVS)

• Init_Synch ~ Override Initial Default Synch
  – Parallel Data Object Sets “synch” Required

• Run_Synch ~ Optimize Run-Time Synch
  – Maintain Synch with All Parallel Tasks, versus
  – Just Maintain Synch with Relevant Tasks
    • Trade Off (Re)Attachment vs. Persistent Overhead…
So What? Who Will Use It?!

• The MxN Port is a nice “Assembly Language.”
  – Performs Fundamental Parallel Data Redistribution
  – Encompasses Connection Types & Synchronization
  – Not Comprehensible to Joe or Jolene Scientist…

• Need Higher-Level Interface:
  – Concept of Component “Users” vs. “Developers”
  – “Implicit” Invocation of MxN Methods in Context?
  – SIDL / BABEL Solution, a la Scott & Gary…
  ➔ Component Developer MxN Run-Time Library!
Unified MxN Internal Structure

- High-Level MxN Run-Time Library
- MxN Data Exchange and Synchronization Interface
- Parallel Data Mapping and Communication
- Data Decomposition Specification
- Data Distribution System
  - Local Data
High-Level MxN Run-Time Library

• “Syntactic Sugar” on MxN Port Interface
  – Driven by Needs of Component Developer

• Hide Details from Casual User
  – Component Methods Handle Internally
  – Resolve Parallel Data Mapping as Needed

• Component’s MxN “Callbacks” Can Be Built Using Standard MxN Port Interface
  – Explicit Developer Code, or SIDL/BABEL…
Example Use of Components

X = getPort("X");
Y = getPort("Y");
Foo = getPort("Foo");

Foo.funct(X.getResult(), Y.getResult());

- Arguments to Foo.funct() Could Be Scalar or Parallel
- Foo.funct() Itself Could Be Serial or Parallel
  → Use MxN Run-Time Library to Resolve…
Serial Foo.funct()

Foo :: funct( X, Y ) {
    \textit{MxN\_Resolve}( X, X\text{scalar} );
    \textit{MxN\_Resolve}( Y, Y\text{scalar} );

    do\_serial\_funct( X\text{scalar}, Y\text{scalar} );
}

\textit{MxN\_Resolve}( in \text{Data} A, out \text{LocalArray} A\text{scalar} ) {
    \text{MxNPort} MxN;
    \text{if} ( \textit{MxN\_isDistributed}( A ) ) \text{then}
        MxN.get\text{Data}( A\text{scalar}, A );  // one-shot
    \text{else} A\text{scalar} = A;
}

Parallel Foo.funct( )

Foo :: funct( X, Y ) {
    int slice = MYRANK * BLKSZ;
    MxN_getRegion( X, myX, {slice, slice + BLKSZ - 1} );
    MxN_getRegion( Y, myY, {slice, slice + BLKSZ - 1} );

    do_local_instance_of_funct( myX, myY );
}

MxN_getRegion( in Data A, out Data myA, in int Bounds[] ) {
    MxNPort MxN;
    myA.decomp.setBounds( Bounds );
    MxN.getData( myA, A );  // one-shot
}

ORNL/Kohl 2001
Persistent Foo.funct( )

Foo :: funct( X ) { 
    VizVar Xviz;
    \texttt{MxN\_coupleData} ( Xviz, X, 10 ); // reduce and interpolate
    \texttt{MxN\_coupleData} ( Xviz, X, 10 ); // evolve and interpolate
    do\_next\_viz\_frame( Xviz );
}

\texttt{MxN\_coupleData} ( in Data myA, in Data A, in int freq ) { 
    MxNPort MxN;
    MxN.setProperty( "freq", freq );
    MxN.getData( myA, A ); // persistent channel
}
Current Challenge…

• Consider Various Component / Data Scenarios
  – Construct Corresponding MxN Library Routines

• Implementation Details Depend on:
  – Base Data Object Specification
  – LocalArray / DistributedArray / Decomposition…
  – MxN Component / Port Implementation

• Summary:
  – Fusion of Explicit and Implicit MxN Interfaces!