Communication overlap – Continued
Heterogeneous processing
Announcements

• Final presentations
  ✷ Friday March 13\textsuperscript{th}, 10:00 AM to 1:00PM
  Note time change.
  ✷ Room 3217, CSE Building (EBU3B)
An alternative way to hide communication

- Reformulate MPI code into a data-driven form
  - Decouple scheduling and communication handling from the application
  - Automatically overlap communication with computation

Scott B. Baden / CSE 262 / UCSD, Wi '15
**Bamboo Programming Model**

- **Olap-regions**: task switching point
  - Data availability is checked at entry
  - Only 1 olap may be active at a time
  - When a task is ready, some olap region’s input conditions have been satisfied

- **Send blocks**
  - Hold send calls only
  - Enable the olap-region

- **Receive blocks**:
  - Hold receive and/or send calls
  - Receive calls are input to olap-region
  - Send calls are output to an olap-region

- Activities in send blocks must be independent of those in receive blocks
- MPI_Wait/MPI_Waitall can reside anywhere within the olap-region

```
1  #pragma bamboo olap
2  {
3      #pragma bamboo send
4          {...}
5      #pragma bamboo receive
6          {... }
7  }

... Computation
.....
```

Scott B. Baden / CSE 262 / UCSD, Wi '15
Results

• Stampede at TACC
  ✷ 102,400 cores; dual socket Sandy Bridge processors
  ✷ K20 GPUs

• Cray XE-6 at NERSC (Hopper)
  ✷ 153,216 cores; dual socket 12-core Magny Cours
  ✷ 4 NUMA nodes per Hopper node, each with 6 cores
  ✷ 3D Toroidal Network

• Cray XC30 at NERSC (Edison)
  ✷ 133,824 cores; dual socket 12-core Ivy Bridge
  ✷ Dragonfly Network
Stencil application performance (Hopper)

- Solve 3D Laplace equation, Dirichlet BCs \( (N=3072^3) \)
  7-point stencil \( \Delta u = 0, \ u=f \) on \( \partial \Omega \)
- Added 4 Bamboo pragmas to a 419 line MPI code

Scott B. Baden / CSE 262 / UCSD, Wi '15
2D Cannon - Weak scaling study

- Communication cost: 11% - 39%
- Bamboo improves MPI-basic 9%-37%
- Bamboo outperforms MPI-olap at scale
Communication Avoiding Matrix Multiplication (Hopper)

- Pathological matrices in Planewave basis methods for \textit{ab-initio} molecular dynamics \((N_g^3 \times N_e)\), For Si: \(N_g = 140, N_e = 2000\)
- Weak scaling study, used OpenMP, 23 pragmas, 337 lines

![TFLOPS/s Graph](image)

- \textbf{Cores}: 4096, 8192, 16384, 32768
- \textbf{Matrix size}: \(N = N_0 = 20608, 2^{1/3}N_0, 2^{2/3}N_0, 2N_0\)
- \textbf{TFLOPS/s}: MPI+OMP, MPI+OMP-olap, Bamboo+OMP, MPI-OMP-nocomm

Scott B. Baden / CSE 262 / UCSD, Wi '15
Virtualization Improves Performance

\[ S_{\text{OMP}} \]

Virtualization factor

Jacobi (MPI+OMP)

Cannon 2.5D (MPI)

\( c=2, \ VF=8 \)
\( c=2, \ VF=4 \)
\( c=2, \ VF=2 \)
\( c=4, \ VF=2 \)
Virtualization Improves Performance

Jacobi (MPI+OMP)

Cannon 2D (MPI)
High Performance Linpack (HPL) on Stampede

- Solve systems of linear equations using LU factorization
- Latency-tolerant *lookahead* code is complicated

<table>
<thead>
<tr>
<th>Finished part of L</th>
<th>D</th>
<th>Uᵢ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lᵢ</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
Aᵢ = Aᵢ - Lᵢ * Uᵢ
\]

- Results
  - Bamboo meets the performance of the highly-optimized version of HPL
  - Uses the far simpler non-lookahead version
  - Task prioritization is crucial
  - Bamboo improves the baseline version of HPL by up to 10%

2048 cores on Stampede
**Bamboo on multiple GPUs**

- **MPI+CUDA programming model**
  - CPU is host and GPU works as a device
  - Host-host with MPI and host-device with CUDA
  - Optimize MPI and CUDA portions separately
- **Need a GPU-aware programming model**
  - Allow a device to transfer data to another device
  - Compiler and runtime system handle the data transfer
  - Hide both host-device and host-host communication automatically
3D Jacobi – Weak Scaling Study

- Results on Stampede
  - Bamboo-GPU outperforms MPI-basic
  - Bamboo-GPU and MPI-olap hide most communication overheads
- Bamboo-GPU improves performance by
  - Hide Host-Host transfer
  - Hide Host-Device transfer
  - Tasks residing in the same GPU send address of the message
Multigrid – Weak Scaling

- A geometric multigrid solver to Helmholtz’s equation [Willams et al. 12]
  - Vcycle: restrict, smooth, solve, interpolate, smooth
  - Smooth: Red-Black Gauss-Seidel
  - DRAM avoiding with the wave-front method

Results:
- Communication cost: 16%-22%
- Bamboo improves the performance by up to 14%
- Communication overlap is effective on levels L0 and L1

| Cores | Comm | Compute | pack/unpack | inter-box copy | Comm/total time at each level |
|-------|------|---------|-------------|----------------|
|       |      |         |             |                | L0  | L1  | L2  | L3  | L4  |
| 2048  | 0.448| 1.725   | 0.384       | 0.191          | 12% | 21% | 36% | 48% | 48% |
| 4096  | 0.476| 1.722   | 0.353       | 0.191          | 12% | 24% | 37% | 56% | 50% |
| 8192  | 0.570| 1.722   | 0.384       | 0.191          | 13% | 27% | 45% | 69% | 63% |
| 16384 | 0.535| 1.726   | 0.386       | 0.192          | 12% | 30% | 48% | 53% | 49% |
| 32768 | 0.646| 1.714   | 0.376       | 0.189          | 17% | 28% | 44% | 63% | 58% |
A GPU-aware programming model

- MPI+CUDA programming model
  - CPU is host and GPU works as a device
  - Host-host with MPI and host-device with CUDA
  - Optimize MPI and CUDA portions separately

- A GPU-aware programming model
  - Allow a device to transfer data to another device
  - Compiler and runtime system handle the data transfer
  - We implemented a GPU-aware runtime system
  - Hide both host-device and host-host communication automatically
3D Jacobi – Weak Scaling

- Results
  - Bamboo-GPU outperforms MPI-basic
  - Bamboo-GPU and MPI-olap hide most communication overheads
- Bamboo-GPU improves performance by
  - Hide Host-Host transfer
  - Hide Host-Device transfer
  - Tasks residing in the same GPU send address of the message
Bamboo Design

- Core message passing
  - Support point-to-point routines
  - Require programmer annotation
  - Employ Tarragon runtime system [Cicotti 06, 11]

- Subcommunicator layer
  - Support MPI_Comm_split
  - No annotation required

- Collectives
  - A framework to translate collectives
  - Implement common collectives
  - No annotation required

- User-defined subprograms
  - A normal MPI program
Bamboo Transformations

• **Outlining**
  - TaskGraph definition: fill various Tarragon methods with input source code blocks

• **MPI Translation**: capture MPI calls and generate calls to Tarragon
  - Some MPI calls removed, e.g. Barrier(), Wait()
  - Conservative static analysis to determine task dependencies

• **Code reordering**: reorder certain code to accommodate Tarragon semantics
for (int iter=0; iter<nIters; iter++) {
    #pragma bamboo olap
    {
        #pragma bamboo send
        {
            ...
        }
        #pragma bamboo receive
        {
            ...
        }
    }
    compute
}

Iter = nIters
Firing & yielding Rule Generation

- Extract source information in all `Recv` and `iRecv` calls of each olap-region, including associated if and for statements
- A task is fireable if and only if it receives messages from all source

Firing rule: while (messageArrival) {
    return test();
}

Yielding rule: yield = ! test();

```cpp
bool test() {
    for (source = 0 to n)
        if (source%2==0)
            if (notArrivedFrom(source))
                return false;
    return true;
}
```
Inter-procedural translation

Bamboo inlines functions that either directly or indirectly hold MPI calls only

```c
void multiGridSolver()
{
    #pragma bamboo olap
    for(int level=0; level<nLevels; level++){
        Send_to_neighbors();
        Receive_from_neighbors();
        Update the data grid
    }
}

void send_to_neighbors()
{
    forall neighbors
        if(neighbor) MPI_Isend(neighbor)
}

void receive_from_neighbors()
{
    forall neighbors
        if(neighbor) MPI_Irecv(neighbor)
}
```

for(cycle=0 to nVcycles
{
    multiGridSolver();
}

main

Scott B. Baden / CSE 262 / UCSD, Wi '15

22
Collective implementation

**Main file**

```c
error = MPI_Barrier(comm);
Bamboo_Barrier(comm);
```

1. Rename collective call
2. Merge library’s AST to main
3. Then inline collective call

```c
comm_0 = comm;
#pragma bamboo
for (int step = 1; step < size; step<<=1) {
    MPI_Send(1 byte to (rank+step)%size, comm_0);
    MPI_Recv(1 byte from (rank-step+size)%size, comm_0);
}
error = SUCCESS;
```

**A collective Library**

```c
int Bamboo_Barrier(MPI_Comm comm){
    #pragma bamboo
    for (int step = 1; step < size; step<<=1) {
        MPI_Send(1 byte to (rank+step)%size, comm);
        MPI_Recv(1 byte from (rank-step+size)%size, comm);
    }
    return SUCCESS;
}
```