Implementing an Open64-based tool for improving the performance of MPI programs

Anthony Danalis  Lori Pollock  Martin Swany  John Cavazos
Problem:
Communication Overhead in Cluster Computing
**Problem:**
Communication Overhead in Cluster Computing

- Inherent to the Application
- Hard or impossible to change
Problem: Communication Overhead in Cluster Computing

- Communication Library & Developer Expertise dependent
- Amendable
## Overall Research Goal

### Requirements:
- Achieve high-performance communication
- Simplify the MPI code developers write

Have your cake + Eat your cake
Overall Research Goal

Requirements:

- Achieve high-performance communication
- Simplify the MPI code developers write

Have your cake + Eat your cake

Automatic cake making machine
Overall Research Goal

Requirements:
- Achieve high-performance communication
- Simplify the MPI code developers write

Proposed Solution:
An **automatic** system that **transforms** simple communication code into efficient code.
Overall Research Goal

Requirements:

- Achieve high-performance communication
- Simplify the MPI code developers write

Proposed Solution:

An automatic system that transforms simple communication code into efficient code.

Side-effect:

Enables legacy parallel MPI applications to perform better, even if written without any knowledge of this system.
Our Framework: ASPhALT*

*Automated System for Parallel Application Transformation

**ASPhALT**

Original Application Code
Fortran, C, C++

Transformed Application

Transformed Code

Linker

Gravel Communication Library

Surveyor
Synthetic Benchmarks for System Discovery

System Knowledge
Throughput, Latency, CPU speed, CPU count, Memory registration cost

Code Transformer
Open64 based compiler transformation phase

**Anthony Danalis**
University of Delaware
Our Framework: ASPhALT*

- **Original Application Code**
  - Fortran, C, C++

- **ASPhALT**
  - **Code Transformer**
    - Open64 based compiler transformation phase

- **Surveyor**
  - Synthetic Benchmarks for System Discovery

- **System Knowledge**
  - Throughput, Latency, CPU speed, CPU count, Memory registration cost

- **Gravel**
  - Communication Library

- **Transformed Application**
  - Transformed Code
  - Linker

---

*Automated System for Parallel Application Transformation

Anthony Danalis | University of Delaware
Integration with Open64

Problems:

**Pros:**
- Portability
- Rapid Prototyping

**Cons:**
- Underutilization of existing:
  - Program analysis
  - Program transformations
Challenges & Opportunities

Gravel enables explicit memory registration & rendezvous handshaking at the application layer

➢ Memory Registration Location
  ✔ Smart placement can reduce registration cost
  ✗ Choosing location requires inter-procedural analysis

➢ Rendezvous Protocol Choice
  ✔ Early handshake initiation can overlap the control messages
  ✔ Advanced rendezvous can reduce control messages & increase overlap
  ✗ Control flow analysis required to map MPI code to Gravel protocol
Memory Registration: context awareness matters

Frequent registering & unregistering message buffers induces significant cost
Gravel rendezvous protocols

Use of wrong rendezvous protocol would lead to deadlock
Mapping MPI to Gravel rendezvous

**Trivial:**
execution order = textual order

```fortran
mpi_irecv()
do i=1,N
    sbuf[ i ] = ...
enddo
mpi_isend( sbuf )
mpi_waitall()
```

**More complex:**
execution order = reverse textual order

```fortran
do i=1,N
    if( i > 1 ) then
        mpi_irecv()
        ...
        mpi_waitall()
    endif
enddo

do j=1,N
    sbuf[ i ][ j ] = ...
enddo

if( i < N ) then
    mpi_isend( sbuf )
endif
endo
```

Example: NAS SP `{x,y,z}_solve.f`
Further Optimizations (future work)

### Before

```plaintext
mpi_irecv()

do i=1,N
    sBuf[i] = ...
enddo

mpi_isend()
mpi_waitall()
```

### After

```plaintext
gravel_post_recv_buffer_rdma()

do T=1, N, K
   do i=T, min(T+K-1, N)
      sBuf[i] = ...
   enddo
   if( T == 1 ) then
      gravel_wait_recv_buffer_rdma()
   endif
   gravel_post_os_put()
enddo
gravel_send_fin(next, ierr)
gavel_waitall()
```
Current & Future Directions

➢ Break MPI collectives into loops of send()/recv()

➢ Use Loop Nest Optimizations to increase overlap

➢ Integrate code into Open64 backend (be)
Questions ?
Gravel: Performance graphs (1)
Gravel: Performance graphs (2)